

ASSEMBLEUR DE L'AMSTRAD

CPC 464 ET 664

MARCEL HENROT

ASSEMBLEUR DE L'AMSTRAD

CPC 464 ET 664

Autres ouvrages relatifs à l'Amstrad

— La découverte de l'Amstrad — Daniel-Jean David
— Exercices en Basic pour Amstrad — Maurice Charbit
— 102 programmes pour Amstrad — Jacques Deconchat
— Super Jeux Amstrad — Jean-François Sehan
— Amstrad en famille — Jean-François Sehan

Pour tout problème rencontré dans les ouvrages P.S.I. vous pouvez nous contacter au numéro ci-dessous :

Numéro Vert/Appel Gratuit en France

16 (05) 21 22 01

(Composer tous les chiffres, même en région parisienne)

ISBN : 2-86595-295-9

MARCEL HENROT

ASSEMBLEUR DE L'AMSTRAD

CPC 464 ET 664



Éditions du P.S.I.
1985

SOMMAIRE

INTRODUCTION ET RAPPELS 1

La programmation en langage machine est la suite naturelle de la programmation en BASIC. L'étude du BASIC de l'Amstrad, dans le guide de l'utilisateur fourni avec l'ordinateur, constitue un préalable recommandé à l'apprentissage du langage machine.

La méthode adoptée consiste à transposer progressivement le langage BASIC en langage machine. La connaissance du premier rend plus facile l'étude du second.

Parmi les avantages de la programmation en codes machine, retenons que :

- la vitesse d'exécution est de loin supérieure à celle du BASIC. C'est primordial pour beaucoup de programmes, et c'est d'ailleurs la vitesse la plus rapide que l'on puisse obtenir ;

- la consommation d'emplacements mémoire est moindre qu'en BASIC. La mémoire disponible permet des programmes plus longs et, ou, plus nombreux ;

- la création de nouvelles commandes ou fonctions, orientées vers les besoins spécifiques de l'utilisateur, est réalisable totalement.

Quelques rappels sont nécessaires avant d'aborder l'étude des codes machine. Ces rappels portent principalement sur les systèmes de numération (le binaire utilisé par tout ordinateur), sur les opérations arithmétiques et logiques réalisées par le microprocesseur, sur l'organisation interne de l'Amstrad et enfin sur le microprocesseur employé : le Z 80. Le lecteur déjà au courant de ces notions, passera à la partie suivante ou au dernier paragraphe de cette partie, qui donne un programme BASIC pour charger les codes machine en mémoire.

1

Les systèmes de numération

L'ordinateur est une machine qui comprend, stocke, et réalise des opérations. Mais il le fait sur des informations qui n'ont que deux états logiques possibles : l'état 0 ou l'état 1, faux ou vrai, bas ou haut. Ceci implique que le **système binaire** est le plus adéquat pour traiter ces informations. Le système binaire est à la base d'un autre système de numération : le **système hexadécimal,** qui rend plus concise la représentation binaire de toute information.

LA NUMÉRATION BINAIRE

Dès maintenant, nous employons le mot bit pour désigner un chiffre binaire, le mot quartet pour désigner un groupe de 4 bits et le mot octet pour désigner un groupe de 8 bits ou de 2 quartets.

Le système binaire, comme le décimal ou l'hexadécimal, est un système positionnel. La position d'un chiffre dans un nombre, soit à gauche, soit à droite de la virgule, confère à ce chiffre une certaine valeur, un certain poids.

Un système de numération positionnel se représente comme suit :

$$... +a_3.B^3 +a_2.B^2 +a_1.B^1 +a_0.B^0 \ , \ +a_{-1}.B^{-1} +a_{-2}.B^{-2} +...$$

où B est la base du système (2 dans le binaire), et où les coefficients a sont pris parmi les chiffres du système (0 et 1 dans le binaire).

Par exemple, le nombre 10011001b peut s'écrire :

$1.2^7 + 0 + 0 + 1.2^4 + 1.2^3 + 0 + 0 + 1.2^0$
$= 128 + 16 + 8 + 1$
$= 153d$

Nous faisons suivre un nombre binaire par b, un nombre décimal par d et un hexadécimal par h, sauf quand il n'y a aucune confusion possible.

Le nombre ci-dessus peut être mis dans un octet, qui est la dimension d'une cellule mémoire de l'Amstrad.

7	6	5	4	3	2	1	0	poids (exposant de 2)
1	0	0	1	1	0	0	1	nombre

Pour préciser le bit (le chiffre) d'un octet, nous employons son poids. Le bit 0 de l'octet ci-dessus est 1, le bit 1 est 0, etc.

LA NUMÉRATION HEXADÉCIMALE

La numération hexadécimale, de base 16, possède 16 chiffres : de 0 à 9, A, B, C, D, E et F (les 6 derniers considérés comme ayant les valeurs 10, 11, 12, 13, 14 et 15 respectivement). Cette numération est plus dense que la binaire ou la décimale : nous l'employons presque constamment.

binaire B=2	décimale B=10	hexadécimale B=16	binaire B=2	décimale B=10	hexadécimale B=16
0	0	0	1010	10	A
1	1	1	1011	11	B
10	2	2	1100	12	C
11	3	3	1101	13	D
100	4	4	1110	14	E
101	5	5	1111	15	F
110	6	6	10000	16	10
111	7	7	10001	17	11
1000	8	8	10010	18	12
1001	9	9	10011	19	13

Tableau 1. – Correspondance entre les numérations

En examinant le tableau 1, vous voyez qu'un quartet correspond à un chiffre hexadécimal. Donc chaque groupe de 4 bits d'un nombre binaire se représente par un chiffre hexadécimal du nombre hexadécimal correspondant.

Par exemple :

$10011001b = 99h = 9.16^1 + 9.16^0 = 144 + 9 = 153d$

Un nombre de 16 bits, qui est le nombre le plus élevé que nous traitons, se représente par 4 chiffres hexadécimaux.

Par exemple :

$1100\ 0000\ 0000\ 0000b = C000h = 12.16^3 = 49152d$

CONVERSION DE BASE

□ *Le passage de base 2 en base 16,* et inversement, est facile. On sépare le nombre binaire en quartets et chaque quartet est converti en un chiffre hexadécimal suivant le tableau 1.

$\underbrace{1100}_{C}\ \underbrace{1000}_{8}$ en base 2, égal en base 16

Il vient naturellement pour le passage inverse :

$\overbrace{1100}^{C}\ \overbrace{1000}^{8}$

□ *Le passage de base 2 ou 16 en base 10* se fait sans difficultés à l'aide de la formule des numérations positionnelles vue plus haut. Nous y avons fait notamment :

10011001b = 153d

□ *Le passage de base 10 en base 2* peut se faire, entre autres, de la façon suivante :

$153 - 128(1.2^7) = 25 - 16(1.2^4) = 9 - 8(1.2^3) = 1 - 1(1.2^0) = 0$

On retranche chaque fois la puissance de 2 la plus élevée possible, pour arriver finalement à 0. Ici les poids 7, 4, 3 et 0 auront le chiffre 1 et les autres poids, le chiffre 0.

□ *Le passage de base 10 en base 16* se fera en deux étapes : conversion de base 10 en base 2, puis de base 2 en base 16. Les conversions en base 16 se font facilement avec l'Amstrad qui possède les routines nécessaires en BASIC :

```
PRINT &C8          affiche 200
PRINT HEX$(200)    affiche C8
```

Tout nombre doit être préfixé par &, dans le BASIC de l'Amstrad, pour que celui-ci reconnaisse un nombre hexadécimal.

2

La représentation des nombres

Quand on manipule des nombres avec une partie fractionnaire, située à droite de la virgule (point pour les Anglais), une erreur de conversion apparaît. Par exemple, pour représenter 0,6d en binaire pur, on peut écrire 0,10011b. Mais même avec 5 chiffres binaires, il reste une erreur d'un peu plus de 1 %.

$$0{,}10011 = 1.2^{-1} + 0 + 0 + 1.2^{-4} + 1.2^{-5}$$
$$0{,}10011 = 0{,}5 + 0{,}0625 + 0{,}03125$$
$$0{,}10011 = 0{,}59375\text{d au lieu de } 0{,}6\text{d}$$

La représentation des nombres en virgule flottante atténue ce genre d'erreur. Mais pour avoir une représentation totalement correcte, on a imaginé le code DCB. Certaines instructions du microprocesseur Z 80A travaillent dans ce code. Pour avoir une précision totale en manipulant des nombres très grands (les comptables notamment), on a recours également au code DCB.

LE CODE DCB

DCB, pour Décimal Codé en Binaire (BCD pour les Anglais), code chaque chiffre décimal en binaire. Il faut 4 bits pour représenter un chiffre décimal. Il y aura donc autant de quartets que de chiffres décimaux. Comme un quartet peut s'arranger de 16 manières différentes et que l'on n'en prend que 10, il reste 6 arrangements inutilisés : c'est un code redondant.

Comme conséquence directe, un octet représente seulement 2 chiffres décimaux, soit un nombre de 0 à 99 quand il est codé en DCB, soit un nombre de 0 à 255 en binaire pur. Il y a donc consommation d'octets mémoire.

décimal	code DCB	décimal	Code DCB
0	0	10	0001 0000
1	1	11	0001 0001
2	10	12	0001 0010
3	11	13	0001 0011
4	100	14	0001 0100
5	101	15	0001 0101
6	110	16	0001 0110
7	111	17	0001 0111
8	1000	18	0001 1000
9	1001	19	0001 1001
		20	0010 0000

Tableau 2. – Codage des nombres en DCB

En conséquence :
1985d s'écrit en DCB : 0001 1001 1000 0101
0,6d s'écrit en DCB : 0,0110

LA VIRGULE FLOTTANTE

L'Amstrad, employé comme calculatrice scientifique, passe en notation exponentielle, de base 10, quand les nombres deviennent trop grands ou trop petits. Quelles que soient leurs représentations à l'affichage, l'Amstrad tient les nombres en mémoire, dans la zone "variables", en virgule flottante, bien sûr en base 2.

Les conversions sont programmées dans la mémoire de l'ordinateur. Notez que l'Amstrad reconnaît également les nombres binaires et hexadécimaux, ainsi que les nombres entiers et réels : vous l'avez remarqué en étudiant le BASIC très étendu de l'Amstrad.
La représentation en virgule flottante se fait en employant la formule :

$$N = m.2^{e-128}$$

où m est la mantisse et e l'exposant.

L'Amstrad utilise 5 octets pour cette représentation. Le premier octet donne la valeur de l'exposant e. Les quatre octets suivants donnent la valeur de la mantisse m dont tous les

bits sont situés à droite de la virgule (la représentation exponentielle, employée par le BASIC, met un chiffre à gauche de la virgule et les autres à droite). Le premier bit significatif, toujours un 1 en binaire, est remplacé par 0 pour un nombre positif et reste 1 pour un nombre négatif.

Dans la zone "variables", l'Amstrad présente les 5 octets dans l'ordre inverse. Si les 5 octets 1 2 3 4 5 représentent un nombre, l'Amstrad les présente ainsi 5 4 3 2 1.

Les conventions ci-dessus amènent les conversions suivantes :

- rétablir l'ordre naturel des 5 octets ;
- écrire les 4 octets de la mantisse en binaire ;
- si le premier bit est 0, le remplacer par 1 ;
- si le premier bit est 1, mettre – devant le nombre ;
- multiplier la mantisse par 2^{e-128}

Pour vérifier ceci, entrons le programme suivant :

```
10 CLS:INPUT "Entrez un nombre":a
20 FOR n=500 TO 550
30 x=PEEK(n):IF x<>193 THEN 60
40 PRINT PEEK(n+2);PEEK(n+3);PEEK(n+4);P
EEK(n+5);PEEK(n+6)
50 END
60 NEXT n
```

Ne vous préoccupez pas de comprendre ce programme, vous ne connaissez pas encore comment est construit un programme BASIC en mémoire. Sachez qu'à la ligne 30, le nombre 193 représente la lettre a dans la zone "variables" et que l'octet suivant est un indicateur.

RUN puis entrons 10, l'écran affiche :

0 0 0 32 132

que nous inversons pour avoir l'ordre naturel :

132 32 0 0 0

Ecrivons la mantisse en binaire :

0,00100000

Le premier chiffre est 0, donc c'est un nombre positif, remplaçons-le par 1 :

0,10100000

Multiplions par $2^{132-128}$ soit 2^4.

Pour multiplier par 2 en binaire, soit la base, on décale le nombre d'un rang vers la gauche (comme pour multiplier par 10 en décimal), il vient :

1010b = 10d

RUN puis entrons 512, l'écran affiche :

0 0 0 0 138

Rétablissons l'ordre :

138 0 0 0 0

Ecrivons la mantisse :

0,00000000

c'est un nombre positif, d'où :

0,10000000

Multiplions par 2^{10}, d'où :

1000000000b = 200h = 2.16^2 = 512d

RUN puis entrons −512, l'écran affiche :

0 0 0 128 138

ou

138 128 0 0 0

la mantisse est :

0,10000000

d'où nombre négatif −512d.

Exercez-vous avec d'autres nombres, si vous voulez approfondir la connaissance de la virgule flottante. Ce n'est pas nécessaire pour poursuivre la lecture de ce livre.

3

Les opérations

LOGIQUES

Le microprocesseur Z 80A effectue les 5 opérations logiques suivantes, et toujours sur un octet.

□ *Le complément restreint*

C'est le complément à B-1 (B étant la base) effectué bit à bit : donc le complément à 1 en binaire. Le complément restreint de A s'écrit A (lire A barre).

Exemple :

si A = 10001000
alors $\overline{A}$ = 01110111

□ *Le complément vrai*

C'est le complément à B ou, plus simplement, c'est le complément restreint plus 1, ou encore, c'est le résultat de l'opération 0 − A.

Dans l'exemple précédent, c'est :

01110111 + 1 = 01111000

□ *L'opération ET (AND)*

Cette opération s'effectue sur deux opérandes, bit à bit. On écrit A ∧ B (lire A et B).

Exemple :

A=10101010
B=01011111
∧ B=00001010

Il suffit d'appliquer les règles suivantes pour chaque bit :

0 ∧ 0=0
0 ∧ 1=0
1 ∧ 0=0
1 ∧ 1=1

□ *L'opération OU (OR)*

Elle s'effectue également bit à bit sur deux opérandes. On écrit A ∨ B (lire A ou B).

Exemple :

A=10101010
B=01011111
A ∨ B=11111111

en appliquant les règles suivantes pour chaque bit :

0 ∨ 0=0
0 ∨ 1=1
1 ∨ 0=1
1 ∨ 1=1

□ *L'opération OU exclusif (XOR)*

Elle s'effectue aussi bit à bit sur deux opérandes. On écrit A ∀ B (lire A ou exclusif B).

Exemple :

A=10101010
B=01011110
A ∀ B=11110100

en appliquant les règles :

0 ∀ 0=0
0 ∀ 1=1
1 ∀ 0=1
1 ∀ 1=0

ARITHMÉTIQUES

Le microprocesseur Z 80A effectue l'addition et la soustraction. Cette dernière opération se transforme en addition grâce à la représentation des nombres négatifs, appelés nombres avec signe ou encore nombres signés (tableau 3).

binaire B=2	décimal B=10	hexadécimal B=16	binaire B=2	décimal B=10	hexadécimal B=16
10000000	−128	80	00000000	0	0
10000001	−127	81	00000001	1	1
10000010	−126	82	00000010	2	2
10000011	−125	83	"	"	"
"	"	"	"	"	"
"	"	"	01111101	125	7D
11111110	−2	FE	01111110	126	7E
11111111	−1	FF	01111111	127	7F

Tableau 3. – Représentation des nombres négatifs

La base étant B, on appelle complément restreint d'un nombre, son complément à B−1 ; et complément vrai, son complément à B.

Ceci étant défini, pour représenter un nombre algébrique (avec signe), on prend la convention suivante : le bit de poids le plus haut (bit 7 pour l'octet ; bit 15 pour l'octet double) sera un bit de signe c'est-à-dire 0 pour un nombre positif et 1 pour un négatif. Ensuite, chaque nombre négatif sera représenté comme le complément vrai du nombre positif correspondant. Un octet pourra représenter un nombre de −128 à 127, et 2 octets consécutifs un nombre de −32768 à 32767.

Le BASIC de l'Amstrad vous donne automatiquement le nombre avec signe pour les nombres qui tiennent en 2 octets.

```
PRINT &7fff      affiche  32767
PRINT &8000      affiche -32768
```

puisque le bit 15 est à 1.

Notez que, pour avoir les nombres avec signe d'un octet, il suffit d'ignorer l'octet haut, ainsi :

```
PRINT HEX$(-2)   affiche  FFFE
```

et l'octet bas FE donne le nombre avec signe en hexa.

Les registres du microprocesseur contiennent 8 bits ; certains peuvent se coupler pour contenir 16 bits. De ce fait, les additions et soustractions se font modulo 256 dans un cas, et modulo 65536 dans l'autre cas. Nous verrons que l'indicateur d'état C signalera si le résultat est modulo. (Quand on fait la division entière de A par B on obtient un quotient Q et un reste R. On a par définition A=B*Q+R. R est le reste de la division entière de A par B et vaut A modulo B). En se référant au système décimal, le compteur kilométrique journalier de votre voiture est modulo 1000 puisqu'il n'affiche que 3 chiffres décimaux. Etant à 999, au kilomètre suivant il va à 000 modulo 1000.

4

Le système Amstrad

La figure 1, page suivante, montre l'organisation interne de l'Amstrad. Nous y trouvons différents dispositifs.

□ *Le microprocesseur Z 80A,* piloté par une horloge à cristal de 4 MHz. Son instruction la plus courte dure 4 périodes d'horloge, soit 1 microseconde, et la plus longue 23 périodes, soit 5,75 microsecondes. Le paragraphe suivant lui est consacré.

□ *Le circuit ULA,* qui contient la logique Amstrad, des portes, des registres et une petite RAM. Ce circuit contrôle les couleurs et les modes d'écran et envoie les signaux vidéo vers le moniteur. Il contrôle également l'usage des mémoires mortes.

□ *La mémoire vive,* ou RAM se compose de 64 K. d'octets. Un K (abréviation de kilo) signifie, en informatique, 2^{10} ou 1024 emplacements mémoire. Chaque emplacement mémoire contient 8 bits et possède une adresse dont la numérotation va de 0 à 65535d, FFFFh. Pour indiquer le contenu d'un emplacement mémoire, nous employons des parenthèses autour de l'adresse. Ainsi (28000)=128 signifie que l'octet d'adresse 28000 a 128 comme valeur. Nous employons également des parenthèses doubles pour signifier contenu du contenu de. Ainsi ((ADR)) donne l'adresse x=(ADR) de la variable ADR et le contenu de cette adresse est (x)=65d (par exemple).

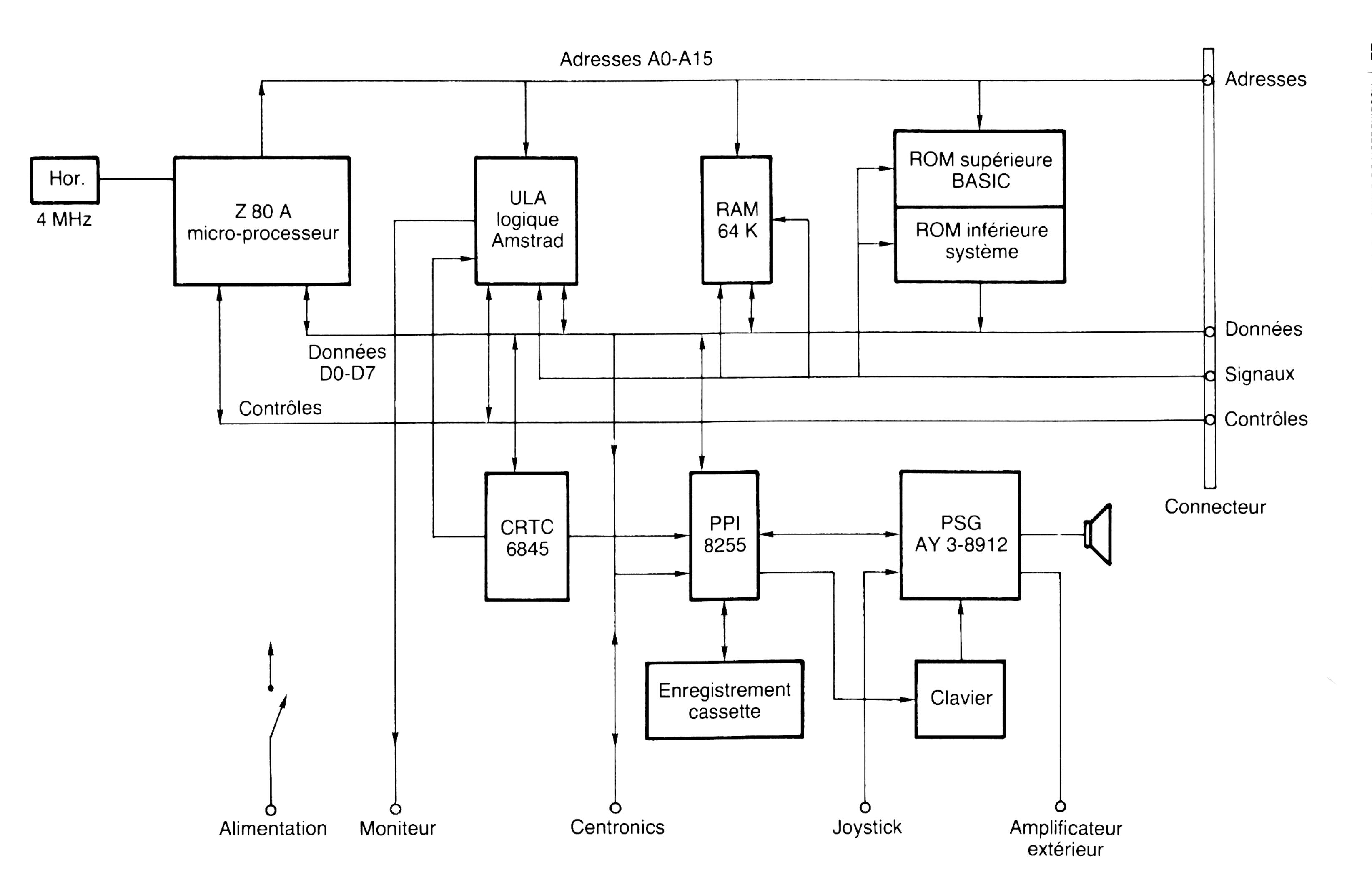


Figure 1. – Synoptique de l'Amstrad CPC 464

- *La mémoire morte* ou ROM se compose de 32 K. d'octets et se divise en deux. La ROM supérieure de 16 K. se situe aux adresses 49152d – C000h à 65535d – FFFFh, et la ROM inférieure, de 16 K. également, aux adresses 0 à 16383d – 3FFFh. Nous voyons qu'on emploie les mêmes adresses que pour deux fois 16 K. de la RAM. C'est l'ULA qui rend les 2 ROM disponibles ou indisponibles, selon l'usage demandé. Ceci est totalement transparent pour l'utilisateur qui programme en BASIC. Pour nous qui désirons programmer en codes machine, nous devrons faire la commutation adéquate. La ROM supérieure contient le système BASIC et la ROM inférieure gère les différents dispositifs du système.

- *Le circuit 6845* contrôle le tube à rayons cathodiques du moniteur. Il envoie à celui-ci, par l'intermédiaire de l'ULA, les signaux de synchronisation adéquats. En ce qui nous concerne, il est programmé pour que ces signaux soient conformes au standard européen 625 E.

- *Le circuit 8255* sert d'interface parallèle pour les différents périphériques. Ce circuit possède trois ports au moyen desquels il gère l'enregistreur à cassette et le circuit qui génère les sons, ce dernier servant de port au clavier. Ce circuit permet également le raccordement d'une imprimante parallèle type "centronics".

- *Le circuit AY3-8912* possède trois canaux de sons. Il peut synthétiser toutes sortes de sons grâce à ses nombreuses options. Ce circuit possède également un port pour saisir l'état du clavier et des joystiks qui seraient raccordés.

- *Un connecteur d'extension* où l'on peut raccorder d'autres périphériques tels que disquettes, ROM supplémentaire, système FORTH, LOGO, etc.

5

Le microprocesseur Z 80

La figure 2, ci-contre, donne l'organisation interne de ce puissant microprocesseur.

Nous trouvons 3 bus qui sortent du microprocesseur pour communiquer avec les circuits extérieurs. Un bus est une réunion de conducteurs imprimés sur le circuit, qui véhiculent les informations. Le bus des données comporte 8 conducteurs D0-D7. Les données entrent dans le microprocesseur pour y être traitées puis en sortent pour être stockées ou utilisées dans les circuits externes. Le bus des contrôles comporte 13 conducteurs de noms divers. Ils contrôlent le comportement du microprocesseur : écriture ou lecture, demande d'interruption, etc. Le bus d'adresses comporte 16 conducteurs A0-A15. Un arrangement de 16 bits sur ce bus fournit une adresse mémoire. Les 3 bus sont tamponnés par un registre TMP qui peut prendre 3 états possibles : les deux états logiques 0 ou 1, ou l'état haute impédance. Le microprocesseur met un bus dans cet état lorsqu'il manipule des données internes. Cet état isole le microprocesseur du monde extérieur.

Nous voyons également 3 unités qui effectuent des opérations : une pour incrémenter et décrémenter, une pour les opérations arithmétiques sur 16 bits et une pour les opérations arithmétiques et logiques sur 8 bits.

Nous trouvons ensuite les différents registres. Tous ces registres sont disponibles pour l'utilisateur à l'exception du registre d'instructions INST, des tampons TMP et du multiplexeur MUX qui sélectionne un des registres situés en dessous de lui. Un registre contient une certaine configuration de bits. Quelle que soit la signification de cet arrangement (instruction, nombre, caractère, contrôle, etc.), on peut toujours le considérer comme un nombre binaire pur.

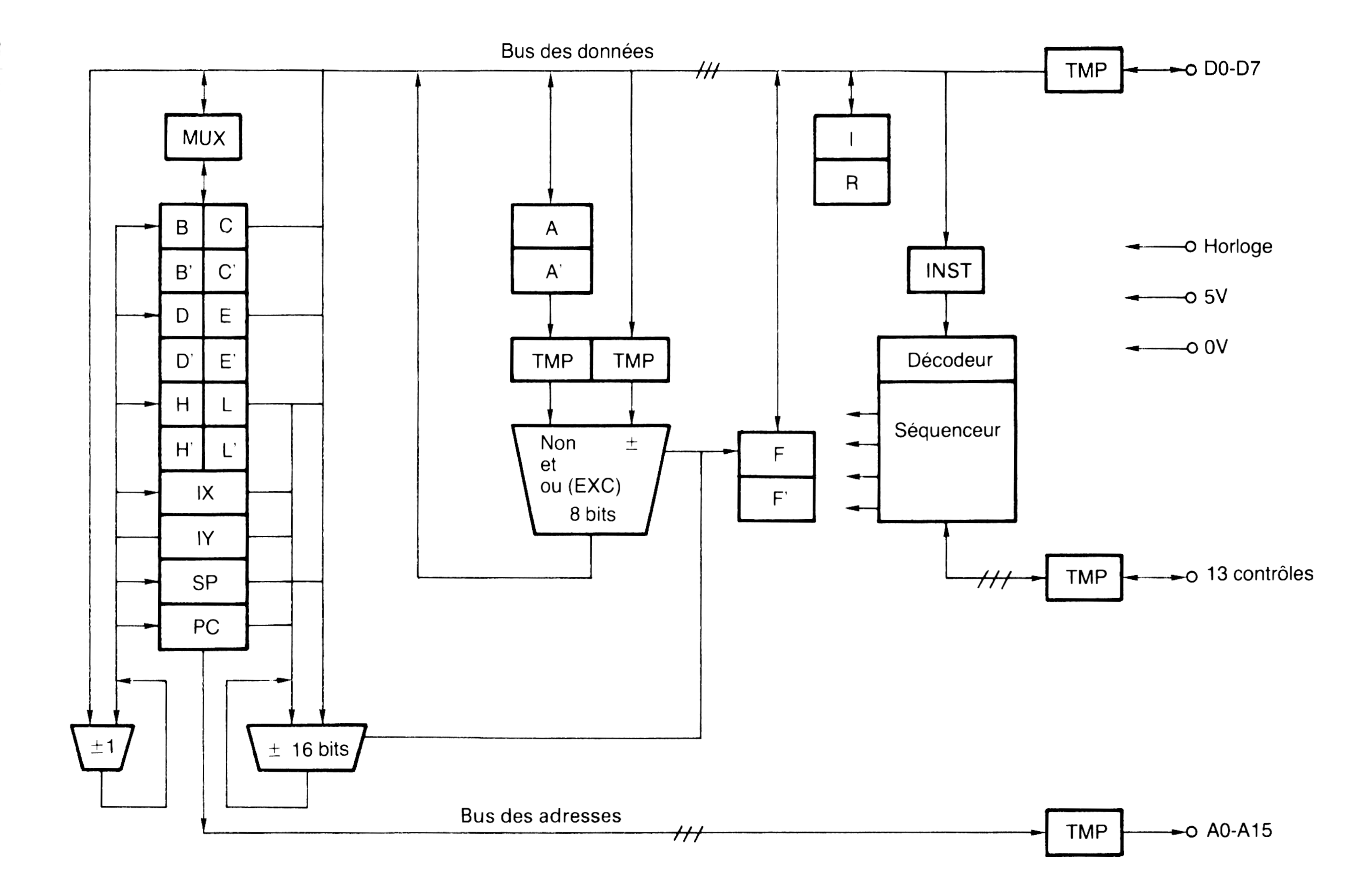


Figure 2. – Synoptique du microprocesseur Z 80A

Disons quelques mots de ces registres :

- *Le registre d'instructions INST* reçoit une instruction, la décode, puis l'exécute en envoyant des signaux aux composants du microprocesseur et vers l'extérieur via le bus des contrôles. Le nombre de cycles d'horloge nécessaire pour exécuter l'instruction dépend de la nature de celle-ci.

- *Le registre I* tient l'octet haut d'une adresse pour un système d'interruption vectorisé, relié à de nombreux périphériques possibles (disquettes, modem, réseau, etc.).

- *Le registre R,* incrémenté régulièrement, sert à rafraîchir des zones de mémoires dynamiques.

- *Les registres B, C, D, E, H et L* sont d'usages multiples. Ils servent d'intermédiaire entre la mémoire extérieure et le microprocesseur pour effectuer des opérations, des transferts, etc. Ces registres peuvent se coupler pour former des registres de 16 bits : BC, DE et HL.

- *Les registres B' à L',* de même que A' et F', doublent les registres existants. Un seul jeu est employé à la fois. Ils servent pour des résultats intermédiaires. Certains programmes de la ROM utilisent le second jeu de registres. Il est déconseillé de s'en servir.

- *Les registres d'index IX et IY* servent pour l'adressage indexé afin de pointer sur une adresse mémoire. Beaucoup d'opérations leur sont permises.

- *Le registre pointeur de pile SP* pointe sur un emplacement mémoire dans une zone de 256 octets (ou plus), réservée pour servir de pile. Cette pile fonctionne suivant le principe LIFO (Last In First Out) : la dernière information entrée dans la pile sera la première à en sortir. La tête de la pile a une adresse assez haute en mémoire, et on empilera aux adresses directement en dessous. Donc, si on pousse un octet dans la pile, SP sera décrémenté et, si on le retire, SP sera incrémenté.

- *Le registre PC* ou compteur de programme, ou encore compteur ordinal, pointe sur l'instruction à exécuter. Après exécution, le PC est incrémenté, plusieurs fois si l'instruction comporte plus d'un octet, pour pointer sur l'instruction suivante.

- *Le registre A* ou accumulateur se trouve sur une des deux entrées de l'unité opératoire 8 bits. Cette unité effectue des opérations sur les deux opérandes présents à ses entrées. Ceci amène un emploi fréquent et obligé de A.

- *Le registre F* ou registre d'états est un registre spécial. Les bits qu'il contient sont des indicateurs d'état. Un ou plusieurs indicateurs sont mis à 1 ou à 0 selon le résultat de l'opération effectuée par les unités opératoires.

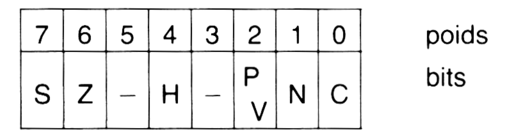


Figure 3. – Configuration du registre F

L'indicateur C est l'indicateur de report (addition) ou de retenue (soustraction). Il est mis à 1 quand le résultat est modulo 256 sur 8 bits ou modulo 65536 sur 16 bits.

L'indicateur P/V a deux fonctions : une fonction de dépassement V pour les opérations arithmétiques et une fonction de parité P pour les opérations logiques. V est mis à 1 quand il y a dépassement de capacité de calcul sur des nombres avec signe. P est mis à 1 quand il y a un nombre pair de bits 1 (ou de bits 0).

L'indicateur Z est mis à 1 quand le résultat de l'opération est nul.

L'indicateur S indique le signe. S est mis à 1 quand le bit le plus significatif du résultat est 1.

Les indicateurs H et N servent pour les opérations qui portent sur des quartets codés en DCB.

6

Les codes machines

Les codes machines représentent des instructions en langage machine que le microprocesseur comprend. Chaque instruction reçue par le microprocesseur, lui fait exécuter une ou plusieurs opérations. Chaque instruction est représentée par une mnémonique. A partir de la partie 2, chaque mnémonique sera expliquée, en donnant son influence sur les indicateurs d'état. Toutes ces explications sont reprises, en annexe, par ordre alphabétique des mnémoniques.

Un octet peut avoir 256 configurations de bits différentes. En affectant un code machine à chaque configuration, on disposerait de 256 instructions. Le Z 80 permet plus de 600 instructions. A cette fin, on emploie les préfixes CB et ED, suivis d'un autre octet pour définir des instructions supplémentaires et les préfixes DD et FD, suivis d'un autre octet pour définir des instructions supplémentaires sur les registres d'index IX et IY. Certaines instructions demandent des nombres qui peuvent aller jusqu'à 65535 et qui sont codés sur un ou deux octets. De ce fait, une instruction pourra comporter de 1 à 4 octets. Le registre PC sera mis à jour automatiquement.

Prenons un exemple : on veut charger 100 dans le registre C. L'instruction C=100 a, pour mnémonique, LD C, 100 et, pour codes machine, OE,64. Cette instruction nécessite 2 octets. La déclaration BASIC 10 LET C=100 nécessite 14 octets en zone programme BASIC plus 7 en zone "variables" après exécution.

Pour représenter les codes machines, nous employons un tableau divisé en 5 colonnes. Une ou plusieurs colonnes peuvent rester vierges.

adresse	codes	étiquette	mnémonique	commentaires
43000	0E,64		LD C,100	

Tableau 4. – Présentation des codes machines

La colonne 1 est écrite en décimal, la 2 en hexadécimal et la 4 dans l'une ou l'autre numération selon le cas.

Les colonnes étiquette et commentaires sont utilisées s'il y a lieu. Seules les deux premières colonnes sont indispensables. Nous devons loger les codes machine de la colonne 2 aux adresses indiquées par la colonne 1.

Signalons qu'il existe des programmes, appelés Assembleur, qui ne nécessitent que l'entrée de la colonne mnémonique. Le programme fait le reste suivant les directives qu'on lui donne et qui sont spécifiques du programme Assembleur du commerce. Nous nous contentons d'entrer les codes machine. Ceux qui possèdent un programme Assembleur feront les translations nécessaires.

PROGRAMME DE CHARGEMENT DES CODES MACHINE

```
10000 CLS
10010 MEMORY 42999:a=HIMEM +1
10020 PRINT "Votre programme commence a"
:a
10030 PRINT "Desirez-vous qu'il commence
 plus haut?":INPUT "Repondez o (pour oui
) ou n (pour non)":e$
10040 IF e$="n" OR e$="N" THEN GOTO 1009
0
10050 INPUT "Si oui, quelle adresse dema
ndez-vous":adr
10060 IF adr<a OR adr>43903 THEN 10070 E
LSE 10080
10070 PRINT "Cette adresse est mauvaise.
":GOTO 10030
10080 a=adr:PRINT "L'adresse de debut es
t":a
10090 PRINT:PRINT "Entrez vos codes mach
ine par groupe de 2"
10100 PRINT "Poussez sur une touche"
10110 IF INKEY$="" THEN 10110 ELSE CLS
10120 LOCATE 1,25:PRINT STRING$(40," ")
10130 LOCATE 1,1:PRINT "Adresse","Code"
10140 LOCATE 1,25:INPUT "Entrez 2 chiffr
es hexa.";a$
```

```
10150 IF a$="l" OR a$="L" THEN 10260
10160 IF a$="e" OR a$="E" THEN 10280
10170 IF a$="s" OR a$="S" THEN STOP
10180 IF LEN(a$)<>2 THEN 10120
10190 a$=UPPER$(a$)
10200 IF LEFT$(a$,1)>"F" OR RIGHT$(a$,1)
>"F" THEN 10120
10210 b=VAL("&"+a$)
10220 POKE a,b
10230 LOCATE 1,23: PRINT a,a$
10240 a=a+1:IF a>43902 THEN STOP
10250 GOTO 10120
10260 b=PEEK(a):a$=HEX$(b)
10270 GOTO 10230
10280 a=a-1
10290 GOTO 10120
```

Les déclarations ont été chiffrées à partir de 10000 pour que vous puissiez mettre votre programme BASIC d'exploitation plus bas. Après la mise au point, vous pouvez l'effacer ou le laisser mais alors, ajoutez la ligne 9999 END pour éviter que l'ordinateur exécute ce programme à la suite du vôtre. Faites également attention si vous employez la commande RENUM pour mettre votre BASIC au point.

Ce programme possède les facilités suivantes. Vous entrez vos codes à partir de 43000 ; si vous voulez une adresse plus haute, il suffit de le signaler. Si vous entrez 1, 3 ou plus de caractères, il n'arrive rien. Il en va de même si vous entrez 2 chiffres non hexadécimaux, comme g2 ou 8h.

Pour lister votre programme, entrez l ou L, vous aurez le contenu de l'adresse. Si vous vous êtes trompé, entrez e ou E pour effacer, vous reviendrez à l'octet pénultième. Pour arrêter, entrez s ou S.

LES OPÉRATIONS DE BASE 2

Ayant vu le moyen d'entrer des codes machine (disons C.M. en abrégé) dans la partie précédente, nous examinons tout d'abord comment exploiter nos programmes en C.M.

Nous présentons ensuite les instructions de chargement qui sont les plus importantes et les plus fréquemment utilisées. Ces instructions réalisent les transferts entre mémoire et registres du microprocesseur. Nous devons constamment prélever des données de la mémoire, les introduire dans des registres pour y être traitées, puis stocker les résultats en mémoire.

Pour terminer, nous présentons les opérations d'addition et de soustraction et celles qui s'y rapportent, telles que l'incrémentation et la décrémentation.

1

Exploitation des programmes en C.M.

Lorsqu'un programme a été introduit en mémoire d'adresse x, pour l'exécuter nous employons la commande BASIC : CALL x. Nous avons besoin d'au moins une commande BASIC pour appeler et exécuter notre programme en C.M.

Pour que l'ordinateur sache où se termine notre programme, la dernière instruction de celui-ci sera une instruction de retour. La mnémonique de cette instruction est RET, et son C.M. est C9. Les opérations que ce C.M. déclenche sont vues plus loin. Voici un programme très court, que vous entrez avec le programme de chargement :

```
43000   C9                                  RET
```

Après l'introduction de C9, entrez s pour arrêter. Redémarrez votre programme de chargement, entrez l pour lister le programme d'un octet, puis s pour arrêter.

Pour exécuter ce programme, faites la commande BASIC :

CALL 43000

A peine appelé, ce programme se termine déjà, et vous avez le message habituel : Ready.

RET signifie, en fait, retour à celui qui a appelé (en anglais RETurn). Ici c'est une commande BASIC qui a appelé, et le retour se fait dans le BASIC.

RET n'a aucune influence sur les indicateurs d'état.

2

Les opérations de chargement

La mnémonique LD, qui signifie charger (en anglais LoaD), n'influence pas les indicateurs d'état, à l'exception de LD A,I et LD A,R qui agissent sur les indicateurs S et Z.

CHARGEMENT DE REGISTRES

□ *Chargement d'un registre par un autre registre*

78	LD A, B	LET A=B

qui signifie : charger B dans A, comme le montre le BASIC correspondant. Les deux arguments sont séparés par une virgule : c'est toujours ce qui est à gauche de la virgule qui change de valeur.

Comme il y a 7 registres simples, il existe 49 C.M. pour ces opérations (voir en annexe).

Pour les registres doubles, il n'y en a que 3 :

F9	LD SP,HL	LET SP=HL
DD,F9	LD SP,IX	SP=IX
FD,F9	LD SP,IY	SP=IY

Une mnémonique telle que LD BC, HL n'est pas valable. Il faut écrire :

44	LD B,H
4D	LD C,L

□ *Chargement d'une constante, de 0 à 255, dans un registre*

3E,nn	LD A,n	LET A=n

Les 7 registres simples peuvent être chargés par une constante. Cette constante n est représentée par nn dans les C.M. pour rappeler qu'il faut 2 chiffres hexadécimaux.

□ *Chargement d'une constante, de 0 à 65535, dans un registre*

01,mm,nn	LD BC,nm	LET BC=nm

Les 3 registres couplés et les 3 registres doubles peuvent être chargés. Si on considère nm comme deux octets dans le BASIC équivalent, il vient LET BC=256*n+m.

Attention : Vous voyez, dans les C.M., que les 2 chiffres hexadécimaux mm représentent l'octet bas et nn l'octet haut. Il faut toujours entrer l'octet bas puis l'octet haut. Ainsi :

11,20,00	LD DE,0020	DE=32
21,00,C0	LD HL,C000	HL=49152

car C0=192d et comme octet haut 192*256=49152 (début du fichier d'affichage).

□ *Chargement d'un registre simple par le contenu d'une adresse*

0A	LD A,(BC)	LET A=PEEK(BC)
1A	LD A,(DE)	
3A,mm,nn	LD A,(nm)	

BC, DE et nm représentent des adresses en mémoire. C'est le contenu de la cellule adressée qui est chargé dans A, comme le montrent les mnémoniques. Dans la correspondance BASIC, les parenthèses sont nécessaires pour la fonction PEEK. Ces trois instructions ne peuvent se faire qu'avec A.

Par contre, les instructions :

7E	LD A,(HL)	A=PEEK(HL)
DD,7E,dd	LD A,(IX+d)	
FD,7E,dd	LD A,(IY+d)	

peuvent se faire avec n'importe quel autre registre simple.

Toutes les instructions qui concernent les registres d'index sont identiques, à l'exception du premier octet qui est DD pour IX et FD pour IY.

Ici, nous voyons pour la première fois l'indexation possible avec ces registres. Nous avons la faculté d'ajouter un déplacement d. Celui-ci est considéré par l'ordinateur comme un nombre avec signe (partie 1). Pour d allant de 0 à 7F, le déplacement est positif, et pour d allant de 80 à FF, le déplacement est négatif. Si vous avez une table de 200 éléments qui commence à l'adresse 48000 et que vous voulez y accéder avec le registre IX, chargez IX, avec la valeur 48100, alors :

```
(IX+10)=(48110)
(IX+254)=(48098)
```

Il n'y a que A que l'on puisse charger avec le contenu d'une adresse spécifiée par une constante. LD B,(nm) n'est pas valable. Il faut alors :

```
21,mm,nn        LD HL,nm
46              LD B,(HL)
```

ou bien :

```
3A,mm,nn        LD A,(nm)
47              LD B,A
```

Vous prendrez l'une ou l'autre manière, selon que A ou HL est disponible à ce moment-là.

□ *Chargement d'un registre double par le contenu de 2 adresses successives*

```
ED,4B,mm,nn     LD BC,(nm)
```

qui signifie :

```
LET BC=PEEK(nm)+256*PEEK(nm+1)
```

ou

```
LET C=PEEK(nm)
LET B=PEEK(nm+1)
```

```
ED,5B,mm,nn     LD DE,(nm)
ED,6B,mm,nn     LD HL,(nm)
ED,7B,mm,nn     LD SP,(nm)
   2A,mm,nn     LD HL,(nm)
DD,2A,mm,nn     LD IX,(nm)
FD,2A,mm,nn     LD IY,(nm)
```

Il y a 2 C.M. possibles pour charger HL. Nous prendrons celui de 3 octets. Il y a quelques redondances semblables dans le jeu d'instructions du Z 80.

Attention : Quand on charge un registre double, on effectue deux opérations :

RRbas=(nm)
RRhaut=(nm+1)

on charge 2 octets, comme le montre le BASIC correspondant.

CHARGEMENT DANS LA MÉMOIRE

□ *Chargement d'une cellule mémoire par un registre simple*

02	LD (BC),A	POKE BC,A
12	LD (DE),A	
32,mm,nn	LD (nm),A	

Ici aussi BC, DE et nm représentent des adresses mémoire ; c'est l'octet de ces cellules qui est chargé par A. Ces trois instructions ne se font qu'avec A. Par contre, les instructions :

77	LD (HL),A	POKE HL,A
DD,77,dd	LD (IX+d),A	
FD,77,dd	LD (IY+d),A	

peuvent se faire avec n'importe quel registre simple.

Pour réaliser LD (nm),B qui n'existe pas, on fait :

78	LD A,B
32,mm,nn	LD (nm),A

□ *Chargement d'une constante dans un octet mémoire*

36,nn	LD (HL),n	POKE HL,n
DD,36,dd,nn	LD (IX+d),n	
FD,36,dd,nn	LD (IY+d),n	

Dans les C.M. des registres d'index, d vient toujours en troisième position. Si on veut LD (IX),n on fait d=0

DD,36,00,nn	LD (IX+0),n

□ *Chargement de 2 octets successifs mémoire par un registre double*

ED,43,mm,nn LD (nm),BC

qu'on peut traduire par :
POKE nm,C
POKE nm+1,B
ou par :
POKE nm,BC−FIX(BC/256)*256
POKE nm+1,FIX(BC/256)

Voyez les six autres instructions possibles en annexe. Il y a aussi deux possibilités pour HL.

Attention : Ces C.M. réalisent deux opérations :
(nm)=RRbas
(nm+1)=RRhaut

Exemple 1 : Pour afficher sur l'écran.

En partie 1, nous avons vu que le système Amstrad était situé dans les 16 K. de ROM inférieure. Ce système, très élaboré, possède, notamment, des nombreuses routines que l'interpréteur BASIC utilise. Nous ferons de même. Pour appeler l'une de ces routines, nous disposons d'une instruction semblable à celle du BASIC :

CD,qq,pp CALLpq

qui appelle la routine d'adresse pq. Cette routine pq se termine par un RET et le retour se fait dans notre programme en C.M.

Pour envoyer un caractère sur l'écran, on emploie la routine d'adresse BB5A. Avant l'appel, le registre A doit contenir le code ASCII du caractère à afficher. Au retour aucun registre n'est modifié.

Ecrivons un programme pour afficher SALUT :

```
43000  3E,53      LD A,83d     :code de S
43002  CD,5A,BB   CALL BB5A    :affiche S
43005  3E,41      LD A,65d     :code de A
43007  CD,5A,BB   CALL BB5A
43010  3E,4C      LD A,76      :code de L
43012  CD,5A,BB   CALL BB5A
43015  3E,55      LD A,85      :code de U
43017  CD,5A,BB   CALL BB5A
43020  3E,54      LD A,84      :code de T
43022  CD,5A,BB   CALL BB5A
43025  C9         RET
```

Faites la commande :

CLS:CALL 43000

et vous voyez le mot SALUT affiché en haut à gauche.

Si vous refaites la commande CALL 43000, le mot s'affiche plus bas. Le fait d'employer une routine du système nous dispense d'effectuer la mise à jour de la position d'affichage ainsi que la manipulation d'autres variables système.
Les codes ASCII des caractères affichables, de 32d à 126d, se trouvent dans le guide de l'utilisateur, en annexe 3.

Exemple 2 : Pour afficher n'importe où sur l'écran.
Pour cela, on emploie la routine système d'adresse BB75.

Avant l'appel, H contient le n° de colonne, de 1 à 40, et L le numéro de ligne, de 1 à 25. Au retour, les registres A, H, L et F sont altérés.

Affichons SALUT, en haut à droite. Nous pourrions recommencer tout le programme qui consiste à déplacer le curseur puis à afficher ; procédons autrement. Faites tourner le programme de chargement ; répondez o (pour oui) à la question quelle adresse ? répondez 43026 ; puis entrez :

```
43026  21,01,24              LD HL,2401h     ;H=36 et L=1
43029  CD,75,BB              CALL BB75       ;position affichage
43032  C9                    RET
```

Faites la commande :

CLS:CALL 43026:CALL 43000

et vous voyez SALUT en haut à droite

Quand nous connaîtrons plus de codes, nous ferons des programmes plus concis pour afficher un texte sur l'écran.

Exercice 1 : a) Ecrivez une ligne BASIC pour afficher SALUT en haut à gauche *et* en haut à droite.
b) Ecrivez une ligne BASIC pour afficher SALUT en haut à droite *et* au début de la deuxième ligne d'écran.

Exercice 2 : Ecrivez un programme en C.M. pour afficher Ami au milieu de l'écran.

3

Les opérations arithmétiques

LES ADDITIONS

□ *Addition d'un octet :*

C6,nn	ADD A,n	LET A=A+n
80	ADD A,B	LET A=A+B

On peut ajouter n'importe quel registre simple à A.

86	ADD A,(HL)	LET A=A+PEEK(HL)
DD,86,dd	ADD A,(IX+d)	
FD,86,dd	ADD A,(IY+d)	

On ne peut additionner un octet (constante, registre, mémoire) qu'à A. Pour réaliser ADD B,n qui n'existe pas, on fait :

```
LD A,n
ADD A,B
LD B,A
```

ou

```
LD A,B
ADD A,n
LD B,A
```

Toutes ces instructions d'addition influencent les indicateurs S, Z, V et C.
C'est le résultat de l'addition qui influence ces indicateurs, ainsi :

C6,00 ADD A,0 LET A=A+0

laisse A inchangé, mais C et V=0 ; et S et Z ne changent pas (s'ils étaient mis par la valeur de A).

C'est la seule instruction, vue jusqu'ici, qui fasse C_i=0. Nous mettons C_i pour désigner l'indicateur C, afin de ne pas le confondre avec le registre C.

ADD A,B veut dire LET A=A+B
mais aussi IF (A+B)>255 THEN LET C_i=1
mais aussi IF (A+B)=256 THEN LET Z=1
mais aussi IF (A+B)> 127 THEN LET S=1
parce que le résultat est modulo 256.

Ceci montre que la traduction en BASIC est toujours circonstanciée. Un C.M. peut faire plusieurs commandes BASIC, et une commande BASIC peut nécessiter plusieurs C.M.

Exemple 3 : Addition modulo 256.
Entrez le programme suivant :

```
43000  00.00        DEF variable
43002  3E.00        LD A,x
43004  C6.00        ADD A,y          :fait x+y
43006  32.F8.A7     LD (43000),A     :(A7F8),A
43009  C9           RET
```

Entrez le BASIC suivant :

```
10 CLS
20 PRINT "Realisons x+y=s"
30 INPUT "Entrez x":x:POKE 43003,x
40 INPUT "Entrez y":y:POKE 43005,y
50 CALL 43002
60 PRINT:PRINT x;"+";y;"=";PEEK(43000)
70 PRINT:GOTO 30
9999 END
```

RUN puis entrez 52, puis 64, vous avez 116.
Entrez 200, puis 100, vous avez 44 mod 256 soit 300-256.

Entrez les couples que vous voulez, mais toujours des nombres plus petits que 256, sinon vous aurez une erreur BASIC.
Le programme commence à 43002. Les deux premiers octets sont réservés ; DEF n'est pas une mnémonique, il indique que ces deux octets sont réservés pour des variables. Cette façon de procéder permet de les récupérer dans le BASIC. Au chargement, vous pouvez entrer 00; de toute façon la ligne 43006 le modifiera.
Pour voir l'influence sur les indicateurs, surtout sur l'indicateur V qui est souvent mal compris, nous examinerons quatre cas. Mais auparavant, modifions les programmes pour que l'on puisse afficher les indicateurs. Ajoutons des lignes de C.M. (en en supprimant une) :

```
43009   F5           PUSH AF          :sauve AF
43010   C1           POP BC           :C=F
43011   79           LD A,C           :A=F
43012   32,F9,A7     LD (43001),A     :(43001)=F
43015   C9           RET
```

Pour les besoins de notre démonstration, nous employons deux mnémoniques que nous verrons plus loin : PUSH et POP. Sachez que le résultat des deux instructions est que C contient le registre F.

Ajoutons également des lignes BASIC (en en supprimant une) :

```
70 a$=BIN$(PEEK(43001),8)
80 PRINT:PRINT "S=";LEFT$(a$,1)
90 PRINT "Z=";MID$(a$,2,1)
100 PRINT "V=";MID$(a$,6,1)
110 PRINT "C=";RIGHT$(a$,1)
120 PRINT:goto30
```

Examinons quatre cas :

a) entrons 253, puis 3 : la somme est 0 mod. 256
S=0 le résultat est plus petit que 128
Z=1 le résultat est nul
V=0 voir explication plus loin R6=1
C=1 il y a report dans l'addition des bits 7

b) entrons 253, 128 : la somme est 125 mod 256
S=0
Z=0 résultat non nul
V=1 voir plus loin R6=0
C=1 il y a report des bits 7

c) entrons 100,125 : la somme est 225
S=1 le résultat est plus grand que 127
Z=0 non nul
V=1 voir plus loin R6=1
C=0 pas de report des bits 7

d) entrons 60,67 : la somme est 127
S=0
Z=0
V=0 voir plus loin R6=0
C=0

Nous possédons les quatre cas pour expliquer l'indicateur V. Nous vous conseillons fortement d'effectuer les quatre additions ci-dessus en binaire, pour savoir s'il y a un report dans l'addition des bits 6. Nous indiquons R6=1 s'il y en a un.

Par exemple, l'addition a) :

```
C 7 6 5 4 3 2 1 0     bits
1 1 1 1 1 1 1 1       reports
  1 1 1 1 1 1 0 1     253
  0 0 0 0 0 0 1 1     3
-----------------
1 0 0 0 0 0 0 0 0     somme
```

Ici, il y a report dans toutes les colonnes de bit, donc R6=1 et l'addition des bits 7 amènent C=1.

Nous avons appris le OU exclusif dans la partie 1, nous constatons que :

V=R6 ∀ C

quand R6 et C sont différents, V=1 pour indiquer un dépassement dans l'addition des nombres avec signe. En effet, si on considère les nombres ci-dessus comme des nombres avec signe, il vient :

a) −3 + 3 =0	vrai	V=0
b) −3 + −128 =125	faux	V=1
c) 100 + 125 = −31	faux	V=1
d) 60 + 67 = 127	vrai	V=0

C'est vous qui décidez si vos nombres sont avec signe ou non (binaire pur). L'ordinateur vous le signale simplement avec son indicateur V. Cependant il y a des nombres que l'ordinateur considère toujours avec signe : le déplacement d des instructions avec registre d'index, et un nombre que nous verrons dans les sauts relatifs qui représente également un déplacement.

□ *Addition des registres doubles*

09	ADD HL,BC	HL=HL+BC
19	ADD HL,DE	HL=HL+DE

Voir en annexe le reste des C.M.
Ces instructions n'influencent que l'indicateur C.

On ne peut additionner qu'à HL, IX ou IY.

Exercice 3 : Ecrivez un programme en C.M. pour additionner des nombres de 2 octets, de 0 à 65535, modulo 65536. Voici le BASIC d'exploitation :

```
10 CLS
20 PRINT "Realisons x+y=s"
30 INPUT "Entrez x":x:POKE 43003,x-FIX(x
/256)*256:POKE 43004,FIX(x/256)
40 INPUT "Entrez y":y:POKE 43006,y-FIX(y
/256)*256:POKE 43007,FIX(y/256)
50 CALL 43002
60 PRINT:PRINT x;"+";y;"=";PEEK(43000)+2
56*PEEK(43001)
70 PRINT:GOTO 30
9999 END
```

□ *Addition avec l'indicateur C*

CE,nn	ADC A,n	$A=A+n+C_i$
88	ADC A,B	
8E	ADC A,(HL)	
ED,4A	ADC HL,BC	

Avec l'addition de C_i, on ne peut additionner qu'à A ou HL.
Les instructions ADC influencent les indicateurs S, Z, V et C.
Il n'y a pas d'instruction pour additionner une constante à HL. Si on veut faire LET HL=HL+64, alors :

01,40,00	LD BC,64d
09	ADD HL,BC

Si BC est employé pour un autre usage, on fait alors :

7D	LD A,L
C6,40	ADD A,64
6F	LD L,A

```
7C          LD A,H
CE,00       ADC A,0
67          LD H,A
```

Voyez-vous pourquoi les trois dernières instructions sont nécessaires ?
C'est pour additionner le C_i éventuel mis par ADD A,64.
Les instructions ADC permettent les additions en multi précision.

□ *Les incrémentations*

L'incrémentation est l'addition d'une unité.

```
3C          INC A          LET A=A+1
```

de même pour les 6 autres registres simples.

```
34          INC (HL)
```

qui réalise l'instruction BASIC POKE HL,PEEK(HL)+1 même instruction pour (IX+d) et (IY+d).
Les instructions d'incrémentation d'un octet, registre ou mémoire, influencent S, Z, V mais ne modifient pas C_i.

```
03          INC BC         BC=BC+1
```

de même pour les 5 autres registres doubles.
Les instructions d'incrémentation d'un registre double, n'influencent *aucun* indicateur.
Les incrémentations se font, bien sûr, mod 256 ou mod 65536.

LES SOUSTRACTIONS

```
D6,nn       SUB A,n        LET A=A−n
90          SUB A,B        A=A−B
```

On peut soustraire n'importe quel registre de A.

```
96          SUB A,(HL)     A=A−PEEK(HL)
DD,96,dd    SUB A,(IX+d)
FD,96,dd    SUB A,(IY+d)
```

Tous les SUB influencent les indicateurs S, Z, V et C.
On ne peut soustraire que de A.
Il n'y a pas de SUB pour les registres doubles.

SUB A,A fait à la fois A=0 et C_i=0. C'est une façon de mettre C_i à 0 quand on n'a pas besoin de A.

SUB A,B effectue A=A−B, mais aussi :

IF B>A THEN C_i=1
IF B=A THEN Z =1
etc.

□ *Soustraction tenant compte de l'indicateur C*

DE,nn	SBC A,n	A=A−n−C_i
98	SBC A,B	

idem pour les 6 autres registres simples.
Et pour les octets mémoire

9E	SBC A,(HL)	A=A−PEEK(HL)−C_i

idem pour (IX+d) et (IY+d)

ED,42	SBC HL,BC	HL=HL−BC−C_i

idem pour DE, HL et SP.

On ne peut soustraire (avec l'indicateur C) un octet que de A et 2 octets que de HL.

Tous les SBC influencent les 4 indicateurs principaux.
SBC A,A est utile et souvent employé :

si C_i=0 alors A=0 et C_i=0
si C_i=1 alors A=255 (ou −1) et C_i=1

Pour faire SUB HL,BC qui n'existe pas, il faut mettre C_i à 0, puis employer SBC :

C6,00	ADD A,0	;C_i=0
ED,42	SBC HL,BC	

ou :

97	SUB A,A	;C_i=0
ED,42	SBC HL,BC	

qui consomme un octet en moins, mais met A à 0.

Pour faire SBC BC,DE qui n'existe pas, on peut employer des instructions de pile mais, avec nos connaissances actuelles, on fait :

60	LD H,B
69	LD L,C
ED,52	SBC HL,DE
44	LD B,H
4D	LD C,L

Exemple 4 : Soustraction modulo 65536.

Entrez le programme suivant :

```
43000  00,00           DEF variable
43002  21,00,00        LD HL,x
43005  01,00,00        LD BC,y
43008  C6,00           ADD A,0          ;Ci=0
43010  ED,42           SBC HL,BC
43012  22,F8,A7        LD (43000),HL
43015  C9              RET
```

Employons le BASIC de l'exercice 3, en remplaçant le signe + par le signe − dans les lignes 20 et 60.
Essayez quelques soustractions pour vous familiariser :

100 − 101 = 65535 , etc.

□ *Les décrémentations*

Ce sont des soustractions d'une unité.

3D	DEC A	A=A−1

de même pour les 6 autres registres simples.

35	DEC (HL)	POKE HL,PEEK(HL)−1

idem pour les registres d'index.
Les décrémentations d'un octet, registre ou mémoire, influencent les indicateurs S, Z, V et C.

0B	DEC BC	BC=BC−1

idem pour les cinq autres registres doubles.
La décrémentation d'un registre double n'influence *aucun* indicateur.

CHANGEMENT DE LA VALEUR D'UN BIT D'UN OCTET

3F	CCF	$C_i=1-C_i$

c'est le complément restreint de C_i.

37	SCF	$C_i=1$

Ces deux instructions ont un effet obligé sur C_i.
Pour mettre C_i à 0, on peut faire :

SCF
CCF

CB,87 RES 0,A

qui réalise $A_0=0$; donc met le bit 0 de A à 0.

CB,C7 SET 0,A

qui réalise $A_0=1$; donc met le bit 0 de A à 1.
Voir les 160 C.M. de RES et SET en annexe.
SET met à 1 et RES met à 0 le bit désigné à gauche de la virgule, du registre ou de l'octet mémoire désigné à droite de la virgule.

Chaque cas a une traduction différente en BASIC, puisqu'on agit sur un bit. Si on sait que A=0, alors SET 7,A exécute LET A=128. D'autre part, quel que soit A, SET 0,A exécute A=A+1 si A était pair et A=A si A était impair ; A devient ou reste impair. Les instructions de manipulation de bits sont toujours circonstanciées.

Exercice 4 : A l'aide d'une instruction INC, affichez ABCD sur l'écran en haut à gauche

PLUSIEURS OPÉRATIONS DE BASE SIMULTANÉMENT

Nous avons vu des mnémoniques et leurs C.M. associés qui effectuent une ou deux opérations ; en voici quatre qui effectuent quatre opérations et qui, par conséquent, sont très puissants. Quand vous rencontrerez des C.M. semblables, ou leurs mnémoniques, remémorez-vous toutes les opérations qu'ils exécutent, car la suite du programme en tient compte et cette suite pourrait vous paraître obscure.

ED,A8 LDD
qui réalise :

(DE)=(HL) POKE DE,PEEK(HL)
DE =DE−1
HL =HL−1
BC =BC−1

ED,A0 LDI
qui réalise :

(DE)=(HL)
DE =DE+1
HL =HL+1
BC =BC−1

Les trois registres couplés sont utilisés. Ils seront chargés préalablement à des valeurs convenables, sauf si on utilise ces C.M. pour une instruction particulière comme (DE)=(HL) qui correspond à LD (nm),(pq) qui n'existe pas. On peut les employer pour cette opération, quoique LD A,(pq) suivi de LD (nm),A fasse la même chose en consommant 6 octets au lieu de 2 mais sans toucher à BC ni à DE.
Ces 2 instructions agissent sur l'indicateur V qui vient à 0 si BC=0 après exécution, si non V=1. C'est le seul DEC BC qui influence un indicateur.

ED,B8	LDDR
ED,B0	LDIR

exécutent les mêmes opérations que LDD et LDI, mais avec répétition, jusqu'à ce que BC=0. Ces 2 C.M. mettent évidemment V à 0. Ils servent à transférer des blocs d'octets d'un endroit à l'autre de la mémoire. Comme BC tient 16 bits, le nombre d'octets concernés peut être grand.

Exemple 5 : Remplissons l'écran d'octets 255.
Sans entrer dans les détails du fichier d'écran, que nous voyons à la partie suivante, entrez le BASIC suivant :

```
10 FOR n=49152 TO 65535
20 POKE n,255
30 NEXT n
9999 STOP
```

Faites la commande CLS:RUN et vous voyez l'écran devenir rouge, ligne après ligne.
Entrez le programme en C.M. suivant :

```
43000  3E,FF            LD A,255
43002  32,00,C0         LD (49152),A
43005  21,00,C0         LD HL,49152
43008  11,01,C0         LD DE,49153
43011  01,FF,3F         LD BC,16383
43014  ED,B0            LDIR
43016  C9               RET
```

Faites la commande CLS:CALL 43000 ou, mieux, faites CLS puis CALL 43000. Voyez la vitesse d'exécution.

Exemple 6 : Remplissons l'écran de pixels de différentes couleurs.
Normalement, nous n'avons pas accès aux ROM. Cependant le système Amstrad contient

une routine qui nous le permet. Son adresse est B906. Elle rend la ROM inférieure disponible. Il n'y a aucune condition d'entrée. La sortie rend A avec l'état précédent de la ROM. Laissons cela pour le moment.

Transférons les 16 K. de la ROM inférieure sur l'écran :

```
43000   CD.06.B9          CALL B906
43003   21.00.00          LD HL,0
43006   11.00.C0          LD DE.49152
43009   01.FF.3F          LD BC.16383
43012   ED.B0             LDIR
43014   C9                RET
```

Faites CLS:CALL 43000
N'est-ce pas rapide et amusant ?

Faites MODE 0:CALL 43000
En mode 0 il y a seize couleurs. Les deux dernières sont clignotantes.

RÉPONSES AUX EXERCICES

Exercice 1 :

a) CLS:CALL 43000:CALL 43026:CALL 43000
b) CLS:CALL 43026:CALL 43000:CALL 43000

Exercice 2 :

```
43000   21.0D.13          LD HL.130Dh     :H-19 et L-14
43003   CD.75.BB          CALL BB75
43006   3E.41             LD A.65d        :code de A
43008   CD.5A.BB          CALL BB5A
43011   3E.6D             LD A.109        :code de m
43013   CD.5A.BB          CALL BB5A
43016   3E.69             LD A.105        :code de i
43018   CD.5A.BB          CALL BB5A
43021   C9                RET
```

Faites la commande :
CLS:CALL 43000

Si vous voulez afficher Ami sur la vingtième ligne, faites :
POKE 43001,20:CLS:CALL 43000

Exercice 3 :

Cet exercice est difficile pour un débutant.
Voici le programme :

```
43000  00,00        DEF variable
43002  21,00,00     LD HL,x
43005  01,00,00     LD BC,y
43008  09           ADD HL,BC         ;x+y
43009  22,F8,A7     LD (43000),HL     ;stocke HL
43012  C9           RET
```

Avec le BASIC écrit dans l'énoncé de cet exercice, entrez des couples de nombres compris entre 0 et 65535.

Exercice 4 :

```
43000  3E,41        LD A,65          ;code de A
43002  CD,5A,BB     CALL BB5A
43005  3C           INC A
43006  CD,5A,BB     CALL BB5A
43009  3C           INC A
43010  CD,5A,BB     CALL BB5A
43013  3C           INC A
43014  CD,5A,BB     CALL BB5A
43017  C9           RET
```

Faites la commande CLS:CALL 43000.

BOUCLES, TESTS, L'ÉCRAN | 3

La partie précédente vous a familiarisé avec les opérations d'écriture et de lecture des registres du microprocesseur et des cellules mémoire. Ces instructions sont constamment employées ; les échanges entre programme et mémoire constituent l'essence de la programmation. Un programme BASIC fait constamment des échanges avec les variables qu'il a créées pour les traiter.

Les opérations arithmétiques vues ensuite vous ont montré leurs influences sur les indicateurs d'état. De nombreuses autres instructions influencent les indicateurs d'état. Grâce aux tests que l'on fait sur ceux-ci, on oriente la suite du programme. On réalise ainsi la rupture de séquence qui est un des concepts fondamentaux de tout ordinateur, comme d'ailleurs de tout programme BASIC.

Cette partie étudie ces instructions sur lesquelles repose la structuration des programmes.
Nous continuons également à montrer des exemples et à proposer des exercices.
Voyons tout d'abord comment est constituée la mémoire d'écran.

1

Mémoire d'écran

La mémoire d'écran se situe aux adresses C000-49152 à FFFF-65535. La ROM supérieure occupe les mêmes adresses. Pour cette raison, les routines du système qui agissent sur l'écran sont dans la ROM inférieure.

La figure 1 donne le plan de cette mémoire. Elle est divisée en huit blocs de 2 K. d'octets chacun :

le bloc 0 débute à C000
le bloc 1 débute à C800
le bloc 2 débute à D000
le bloc 3 débute à D800
le bloc 4 débute à E000
le bloc 5 débute à E800
le bloc 6 débute à F000
le bloc 7 débute à F800

D'autre part, on peut considérer l'écran comme 200 lignes de 80 octets ou, comme 25 rangées de 80 octets ; huit lignes de pixels correspondent à la hauteur d'un caractère.

Le bloc 0 remplit successivement la première ligne de chacune des 25 rangées, comme le montre le BASIC suivant, que vous entrez si vous venez de mettre votre ordinateur en marche. Si non, avant d'entrer le BASIC, faites une remise générale en appuyant sur CTRL-SHIFT-ESC, ou en faisant la commande CALL 0, qui effectue la même chose.

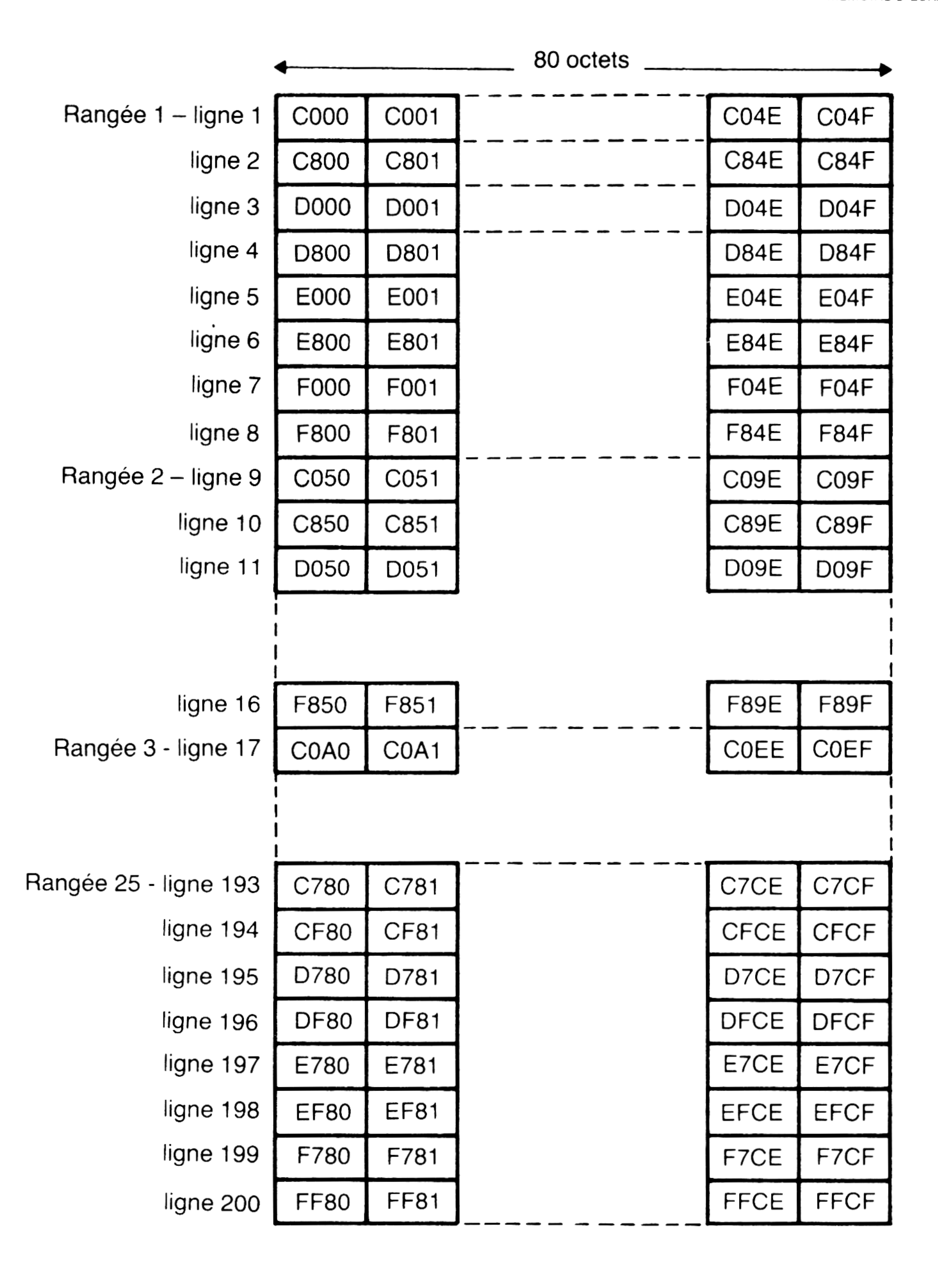


Figure 1. – Organisation de la mémoire d'écran avec offset 0

```
10 CLS
20 FOR n=&C000 TO &C7CF
30 POKE n,255
40 NEXT
```

Le bloc 1 remplit successivement les deuxième lignes de chacune des 25 rangées ; et ainsi de suite.

Pour voir la deuxième ligne de chaque rangée, ajoutez :

```
50 FOR n=&C800 TO &CFCF
60 POKE n,255
70 NEXT
```

Remarquons ceci : 25 fois 80 octets égale 2000 octets, or 2 K. d'octets représentent 2048 octets. Il y a donc 48 octets par bloc, qui semblent inutilisés. Sur la figure 1, ils sont à droite des huit dernières lignes : C7D0 à C7FF, CFD0 à CFFF, ..., FFD0 à FFFF.

L'écran ne démarre pas toujours à C000. On peut ajouter ou retrancher un offset, toujours pair, pour changer le départ de l'écran. L'ordinateur le fait automatiquement quand l'écran doit défiler, en ajoutant une certaine valeur avec signe à l'offset. Par exemple, faites descendre le curseur jusqu'au bas de l'écran en appuyant plusieurs fois sur ENTER ou en laissant appuyée la touche du pavé "curseur" ayant la flèche vers le bas (cette touche est à répétition). Arrivé au bas de l'écran, appuyez plusieurs fois sur ENTER pour faire défiler l'écran plusieurs fois vers le haut, puis faites RUN.

Vous constatez que le remplissage se fait autrement : C000 part ailleurs sur l'écran avec ligne incomplète.

Modifions le BASIC pour utiliser les 48 octets de chaque bloc.

Dans la ligne 20, remplacez C7CF par C7FF.
Dans la ligne 50, remplacez CFCF par CFFF.

Puis RUN, et les lignes précédemment incomplètes se remplissent. Le plan de la mémoire change, mais chaque bloc est continu. Les octets ... C7FE, C7FF, C000, C001, ... sont consécutifs et se trouvent sur la même ligne.

Au départ et à chaque changement de mode d'écran, l'offset est remis à 0. Les trois modes d'écran arrangent les 80 octets de chaque ligne pour produire 20, 40 ou 80 colonnes de caractères, avec des attributs variables selon le mode. Nous verrons ceci plus loin. Vous pouvez tourner le programme BASIC ci-dessus dans les différents modes pour voir diverses couleurs.

2

Les sauts et les boucles

LES SAUTS INCONDITIONNELS

C3,qq,pp	JP pq	GOTO pq

réalise l'opération PC=pq.
Pour rappel, l'octet bas puis l'octet haut de l'adresse doivent être introduits dans cet ordre.

E9	JP HL
DD,E9	JP IX
FD,E9	JP IY

Ces quatre instructions sont des sauts inconditionnels à une adresse, quelle qu'elle soit. JP HL est intéressant, il correspond à un GOTO A, où A est une variable. HL peut résulter d'un calcul, tout comme A dans le BASIC. Ceci équivaut à un GOTO calculé.

18,ee	JR e

réalise l'opération PC=PC + e + 2 où e est un nombre avec signe, tout comme le déplacement d d'une instruction indexée, décrit en partie 2. Les valeurs possibles vont de −126 à 129, en comptant le déplacement à partir de l'adresse de JR e.

43100 18,03	JR 3

Le saut est compté à partir de 43100 et l'instruction suivante sera celle d'adresse 43105. Par conséquent :

JR 0	n'a pas d'effet
JR FE	est une boucle infinie. Vu ?

JR agit comme JP mais dans un espace plus restreint.
L'avantage de JR, qui n'emploie que 2 octets, est de rendre la routine relogeable. Puisque le saut est relatif, aucune adresse absolue n'est donnée ; ce qui n'est pas le cas de JP.

Tous les sauts, inconditionnels ou conditionnels, n'ont aucun effet sur les indicateurs.

LES SAUTS CONDITIONNELS

Commençons par une instruction très souvent employée et puissante parce qu'elle réalise plusieurs opérations.

10,ee DJNZ e

réalise les opérations B=B−1

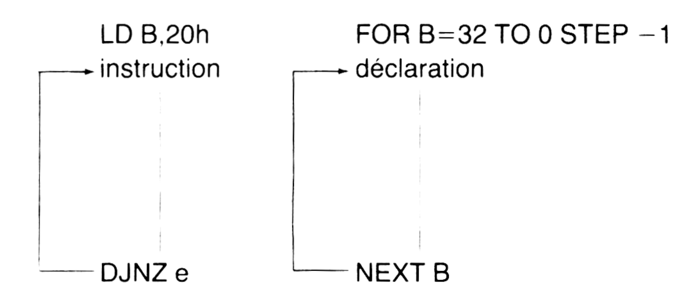
si B≠0 faire PC=PC+e+2
si B=0 poursuivre à PC=PC+2

e est un nombre avec signe. On ajoute 2 comme pour JR e pour tenir compte des deux octets de l'instruction.

DJNZ n'a aucun effet sur les indicateurs, donc DJNZ 0 fait la même chose que DEC B mais sans influencer les indicateurs.

DJNZ sert à contrôler une boucle. Celle-ci est relogeable dans son intégralité. Mais attention : si on ajoute ou retranche une instruction de la boucle, on doit modifier l'argument e en conséquence.

Par exemple : (avec traduction BASIC) :



B sert de compteur. A la fin de la boucle, B=0. Les six instructions sont effectuées 32 fois.

Si l'on veut ajouter 10 à DE, soit ADD DE,10d qui n'existe pas, on fait :

```
LD HL,10
ADD HL,DE
LD D,H
LD E,L
```

sans employer B, ou bien :

```
  LD B,10
┌→INC DE
└─DJNZ FD
```

sans employer HL.

Exemple 1 : Affichons "Bonjour Amstrad" à l'aide d'une boucle.

```
43000  42,6F,6E              DEF Bon
43003  6A,6F,75,72           DEF jour
43007  20,41,6D,73           DEF _Ams (le signe _ pour espace
43011  74,72,61,64           DEF trad
43015  00,00,00,00,00        DEF tampon inemploye
43020  21,F8,A7              LD HL,43000
43023  06,0F                 LD B,15
43025  7E                  ┌→LD A,(HL)
43026  CD,5A,BB            │ CALL BB5A
43029  23                  │ INC HL
43030  10,F9               └─DJNZ -7+2
43032  C9                    RET
```

Faites la commande : CLS:CALL 43020
On charge 15 dans B pour afficher 15 caractères. La boucle, constituée de trois instructions (le chargement de A, l'appel de la routine BB5A, et l'incrémentation de HL), est effectuée 15 fois.

On peut employer un indicateur de fin de message, mais alors la boucle est contrôlée par une comparaison que nous étudions plus loin.

Exercice 1 : Ecrivez un programme pour afficher les 224 caractères affichables, de codes 32 à 255. N'employez pas de tampon.

Poursuivons l'examen des sauts conditionnels :

C2,qq,pp	JP NZ,pq	si Z=0
CA,qq,pp	JP Z,pq	si Z=1
D2,qq,pp	JP NC,pq	si C_i=0

DA,qq,pp	JP C,pq	si C_i=1
E2,qq,pp	JP PO,pq	si P/V=0
EA,qq,pp	JP PE,pq	si P/V=1
F2,qq,pp	JP P,pq	si S=0
FA,qq,pp	JP M,pq	si S=1

Ces instructions effectuent une opération alternative :

- si la condition est vraie, effectue PC=pq
- si la condition est fausse, poursuivre avec PC=PC+3

et pour les sauts conditionnels relatifs, nous avons :

20,ee	JR NZ,e	si Z=0
28,ee	JR Z,e	si Z=1
30,ee	JR NC,e	si C_i=0
38,ee	JR C,e	si C_i=1

qui effectuent :

- si la condition est vraie, faire PC=PC+e+2
- si la condition est fausse, poursuivre à PC=PC+2.

Rappelons que tous les sauts n'influencent aucun indicateur.
Les sauts relatifs sont relogeables, ils testent moins d'indicateurs que les sauts absolus et opèrent dans un espace plus restreint : +129 en avant, −126 en arrière. Notez que les tests sur les indicateurs Z et C sont les plus employés.

Les sauts conditionnels permettent des constructions telles que IF ... THEN ... ELSE ... et FOR ... NEXT. Ils s'emploient, évidemment, après une instruction qui modifie l'indicateur testé.

Par exemple :

```
SUB A,B
JP Z,pq
```

signifient IF A−B=0 THEN GOTO pq.

BOUCLES IMBRIQUÉES

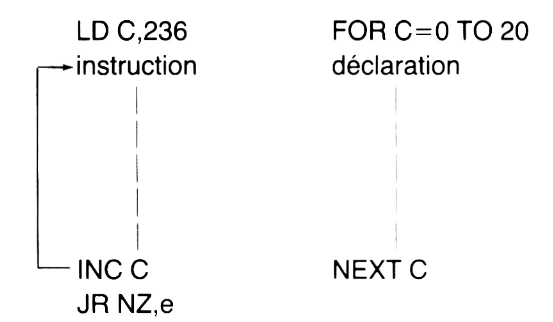
On emploie DJNZ pour une boucle contrôlée par B. Quand il est nécessaire d'avoir plusieurs boucles, on emploie un des autres registres comme compteur.

Par exemple :

```
DEC C
JR NZ,e
```

correspond à un DJNZ e avec C comme compteur.

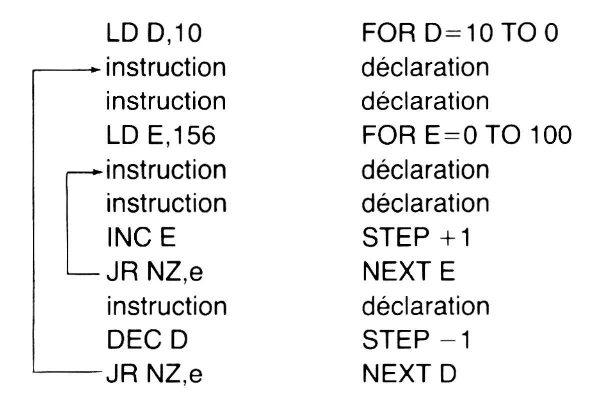
La souplesse d'utilisation est plus grande. Ainsi :



```
  LD C,236          FOR C=0 TO 20
┌→instruction       déclaration
│     |                 |
│     |                 |
└─INC C             NEXT C
  JR NZ,e
```

où la boucle est exécutée avec incrémentation de C.
Quand C a été incrémenté 20 fois, C=0 d'où Z=1 et on quitte la boucle.

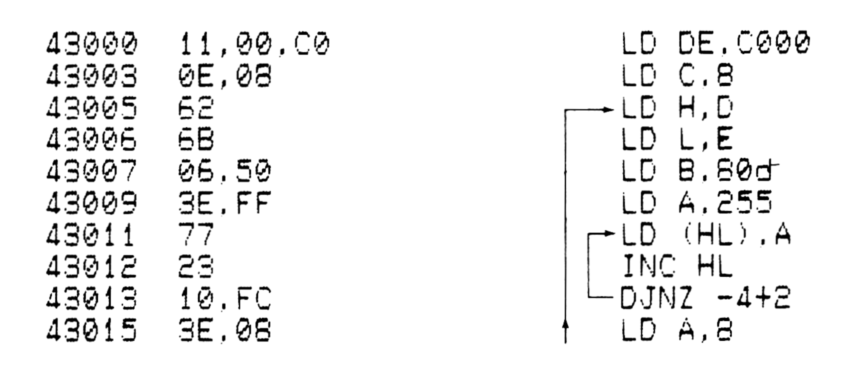
Soient deux boucles imbriquées, contrôlées par D et E :



```
   LD D,10           FOR D=10 TO 0
┌──→instruction      déclaration
│   instruction      déclaration
│   LD E,156         FOR E=0 TO 100
│ ┌→instruction      déclaration
│ │ instruction      déclaration
│ │ INC E            STEP +1
│ └─JR NZ,e          NEXT E
│   instruction      déclaration
│   DEC D            STEP -1
└───JR NZ,e          NEXT D
```

Le BASIC est explicatif ; il n'est pas bien écrit.
La Boucle D est exécutée 10 fois et, à chaque passage, la boucle E est exécutée 100 fois.

Exemple 2 : Ecrivons un programme pour effacer une rangée de caractères où l'on veut.
Tout d'abord, écrivons le programme pour afficher une rangée coloriée.



```
43000  11,00,C0          LD DE.C000
43003  0E,08             LD C.8
43005  62             ┌→ LD H,D
43006  6B             │  LD L,E
43007  06,50          │  LD B.80d
43009  3E,FF          │  LD A.255
43011  77             │┌→LD (HL).A
43012  23             ││ INC HL
43013  10,FC          │└─DJNZ -4+2
43015  3E,08          ↑  LD A,8
```

```
43017  82              ADD A,D
43018  57              LD D,A
43019  0D              DEC C
43020  20,EF        └──JR NZ,-17+2
43022  C9              RET
```

Faites les commandes :
MODE 1:REM pour remettre l'offset à 0 et faire CLS
CALL 43000

La boucle interne 43011-43013 s'effectue 80 fois pour remplir de valeur 255 les octets d'adresses C000 à C04F. Comme HL change dans cette boucle, nous avons mis l'adresse de départ dans DE, puis nous l'avons transférée dans HL aux lignes 43005-43006.

En tenant compte de la figure 1, nous devons ajouter 8 à l'octet haut de l'adresse (registre D), pour remplir la ligne suivante. Les lignes 43015 à 43018 effectuent cette addition en se servant de A. On effectue ensuite le bouclage externe. Nous remplissons ainsi les huit lignes de la première rangée.
Ce programme fonctionne dans les trois modes.

Pour effacer, nous devons introduire 0 dans A pour le transfert dans (HL). Pour ne pas employer le programme de chargement en listant jusqu'à 43010, adresse de l'argument de LD A, modifions cet argument par un POKE.

Faites : POKE 43010,0
puis CALL 43000
et la première rangée s'efface.

Pour agir sur la vingt et unième rangée, modifions l'argument de la ligne 43000. L'adresse de départ est C640h ; soit 20 fois 80 égale 1600-640h que l'on ajoute à C000. La modification est réalisée par deux POKE :

POKE 43001,&40:POKE 43002,&C6

Pour afficher, faisons :

POKE 43010,255

puis :

CALL 43000

Le message Ready n'est pas sur la vingt-deuxième rangée car notre programme ne comporte aucun appel aux routines du système. Leurs appels mettent à jour un tas de variables système pour la gestion convenable de l'ordinateur. Sachez que nos C.M., en tant que tels, n'influencent pas le système Amstrad, sauf si nous l'employons.

Remarque : Introduisons une mnémonique intéressante :

00 NOP

qui n'effectue aucune opération.

Pour faire A=0 dans la ligne 43009, on peut employer SUB A,A de C.M. 97h ; mais on doit également mettre un NOP à l'adresse 43010, pour ne pas devoir modifier les arguments des deux sauts relatifs. Donc :

POKE 43009,&97:POKE 43010,0

changent 2 lignes :

```
43009   97        SUB A,A       ;A=0
43010   00        NOP
```

L'instruction NOP sert à introduire de courts délais (elle dure 1 microseconde) et à corriger des programmes pendant leur mise au point.

Il est rare qu'un programme tourne correctement du premier coup. Il faut modifier des instructions. Si vous avez placé quelques NOP dans votre programme, vous les remplacez par de nouveaux C.M. sans devoir recalculer certaines adresses. De même, quand on supprime des instructions, il est bon de les remplacer par des NOP. Cela évite de recalculer les arguments des sauts relatifs. Nous touchons ici l'un des grands avantages d'un programme Assembleur. Quand vous éditez votre programme source, vous ajoutez ou supprimez les mnémoniques que vous voulez, à l'endroit souhaité. Vous demandez ensuite l'assemblage et votre Assembleur recalcule tous vos sauts. Cependant, travailler en C.M. constitue un très bon exercice puisque c'est la langue maternelle du microprocesseur, et enrichit votre connaissance en profondeur des mécanismes de l'ordinateur. Les programmes dont nous émaillons le livre, ne sont pas trop longs. Si, après lecture de ce livre, vous avez pris goût à la programmation en C.M., nous vous recommandons l'achat d'un programme Assembleur.

Exemple 3 : Ecrivons un programme pour faire défiler la première rangée, d'un caractère à la fois, vers la gauche.
A la demande du programme de chargement, répondez O et entrez 43010.

```
43010   3E,08            LD A,8
43012   11,00,C0         LD DE,C000
43015   ED,53,F9,A7   ┌→LD (43001),DE
43019   32,F8,A7      │  LD (43000),A
43022   62            │  LD H,D
43023   6B            │  LD L,E
43024   23            ↑  INC HL
```

```
43025  01,4F,00        LD BC,004F          :BC=79d
43028  1A              LD A,(DE)
43029  ED,B0           LDIR
43031  00              NOP
43032  12              LD (DE),A
43033  ED,5B,F9,A7     LD DE,(43001)
43037  3E,08           LD A,8
43039  82              ADD A,D
43040  57              LD D,A
43041  3A,F8,A7        LD A,(43000)
43044  3D              DEC A
43045  20,E0           JR NZ,-32+2
43047  C9              RET
```

On charge directement à partir de 43010. Comme le programme de chargement nous a réservé de la place à partir de 43000, nous disposons de 10 octets mémoire comme tampon. On n'en emploie que trois.

En fait, il y a deux boucles dans ce programme. L'une est contrôlée par LDIR et l'autre par A servant de compteur. Comme nous avons besoin de A ailleurs, nous nous servons de l'octet d'adresse 43000 pour son stockage intermédiaire.

Avant LDIR, on définit les 3 registres couplés BC, DE et HL pour le fonctionnement de LDIR et après, on modifie ce qui est nécessaire pour la boucle suivante contrôlée par A.

Au départ, DE=C000 et HL=C001 ; LDIR transfère de (HL) dans (DE). Pour cette raison on fait BC=79d puisqu'on transfère à partir de C001. De plus, la ligne 43028 stocke le premier octet de chaque ligne dans A et, après LDIR, en ligne 43032, on remet cet octet à la fin de la ligne ; ce qui effectue une rotation complète de la rangée.

LDIR modifie DE. Celui-ci définit le début de chacune des 8 lignes de la première rangée. On effectue son stockage intermédiaire aux octets d'adresses 43001 et 43002 pour son traitement.

Voici le programme BASIC d'exploitation :

```
5 MODE 1
10 PRINT "Voici le defilement a gauche p
ar carac."
20 FOR n=1 TO 80
30 CALL 43010
40 FOR t=1 TO 10:NEXT
50 NEXT n
60 END
```

Faites RUN
Le BASIC ralentit le déroulement du programme. Enlevez la ligne 40 et le déroulement est plus rapide. Si on avait programmé la boucle n en C.M., cela irait encore plus vite. Le ralentissement serait nécessaire au moyen de retards programmés que nous étudions plus loin.

Ce programme fonctionne dans les trois modes. Faites :

```
5 MODE 0          puis RUN
5 MODE 2          puis RUN
```

En fait on déplace un octet à la fois avec CALL 43010. Donc en mode 0, on déplace 1/4 de caractère à la fois, en mode 1, 1/2 caractère et en mode 2 tout le caractère. Pour le voir mieux, faites :

```
40 FOR t=1 TO 200:NEXT
```

Au lieu d'avoir une rotation complète, si on désire que les caractères disparaissent en se déplaçant vers la gauche, on charge alors (DE) avec la valeur 0, en faisant :

```
POKE 43031,&97
```

et la ligne 43031 devient :

```
43031     97        SUB A,A             ;A=0
```

Exercice 2 : Ecrivez un programme pour défiler la première rangée vers la droite.
Ce programme est semblable à celui de l'exemple 3 : il y a trois lignes qui changent.

3

Les opérations logiques

Les opérations logiques, vues en partie 1, effectuées par le microprocesseur sont identiques aux fonctions BASIC AND, OR et XOR de l'Amstrad. Si vous êtes habitué à ces fonctions, c'est un avantage certain.

L'OPÉRATION ET

E6,nn AND n

réalise l'opération : $A=A \wedge n$.

AO AND B

réalise : $A=A \wedge B$.
Idem pour les 7 registres simples.

A6 AND (HL)

effectue : $A=A \wedge (HL)$.
Idem pour (IX+d) et (IY+d).

L'opération ET ne se réalise qu'avec le registre A. Cette opération influence les indicateurs S, Z, P et met C_i à 0. L'indicateur P est au même endroit que V (opération arithmétique).

Si l'on veut B=B $\wedge$ C qui n'existe pas, on fait :

```
LD A,B
AND C
LD B,A
```

AND A laisse A inchangé et met $C_i=0$; il est souvent employé pour cette raison.
AND 0 met A=0 ; c'est la même chose que LD A,0, excepté pour les indicateurs.
AND FF laisse A inchangé.

L'opération ET sert souvent à masquer certains bits pour les faire disparaître. Ainsi AND FE rend A pair en faisant disparaître le bit 0 de valeur 1. Certaines adresses doivent être paires dans l'Amstrad.

AND 7 donne à A une valeur comprise entre 0 et 7 : c'est A modulo 8.
AND 6 donne à A la valeur 0, 2, 4 ou 6
AND CO laisse les bits 7 et 6 inchangés, les autres vont à 0.
AND OF laisse le quartet bas inchangé, l'autre va à 0.
Nous vous conseillons d'exécuter ces exemples en binaire, en appliquant les règles de la partie 1, et quel que soit A.

L'OPÉRATION OU

```
F6,nn          OR n
```

réalise l'opération : A=A V n.

```
B0             OR B
```

effectue : A=A V B.
De même pour les sept registres simples.

```
B6             OR (HL)
```

Idem pour (IX+d) et (IY+d).
L'opération OU ne se fait qu'avec A ; elle influence les indicateurs S, Z, P et met C_i à 0.
Comme avec AND, on passe par A pour faire B=B V D

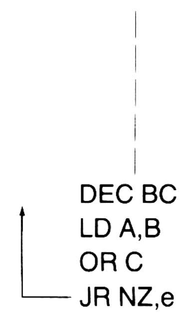
```
LD A,B
OR D
LD B,A
```

Si A=0, alors OR n est le même que LD A,n, sauf pour les indicateurs.
Si A=FF, alors OR n laisse A inchangé.

On se sert souvent de OR pour mettre certains bits à 1, tout en laissant les autres inchangés.

OR C0 met les bits 7 et 6 de A à 1.
OR 80h met le bit 7 de A à 1. Si A était plus petit que 128, A devient A+128 ; si A était plus grand que 127, A ne change pas.

Voyons un autre usage fréquent de OR. Nous avons vu que l'incrémentation ou la décrémentation d'un registre couplé n'influence aucun indicateur. Si on emploie BC pour contrôler une boucle 300 fois, on décrémente BC, et quand BC vient à 0, on veut quitter la boucle. Dans ce cas :



qui effectue un saut relatif tant que BC ≠ 0.

L'OPÉRATION OU EXCLUSIF

EE,nn	XOR n

réalise l'opération : $A=A \forall n$.

A8	XOR B

De même pour les sept registres simples.

AE	XOR (HL)

et aussi avec les registres d'index.
Les OU exclusif ne se font qu'avec A et influencent S, Z, P et C_i qui vient à 0.

La manipulation de XOR est difficile. Appliquez les règles de la partie 1 pour connaître le résultat.

XOR appliqué 2 fois de suite, retourne A inchangé.
XOR A est utile pour faire à la fois A=0 et C_i=0. Notez que cette instruction met aussi S=0, Z=1 et P=1.

LE COMPLÉMENT

2F CPL

effectue l'opération A=A ; c'est le complément restreint de A. Il n'y a pas d'effet sur les indicateurs. On ne peut complémenter que A.

Si l'on veut le complément restreint de BC, on fait :

```
LD A,B
CPL
LD B,A
LD A,C
CPL
LD C,A
```

et si on veut le complément vrai de BC, on ajoute : INC BC

ED,44 NEG

exécute l'opération A=0−A : c'est le complément vrai de A.

NEG agit sur les indicateurs comme suit :

S et Z sont mis suivant le résultat ; P=1 si A contenait 80h avant l'opération ; C_i=1 si A contenait 0.

Si A=80h, NEG fait A=80h.
Si A=1, NEG fait A=255 ou −1 (nombre avec signe).
Si A=9, NEG fait A=247 ou −9 (avec signe).

CPL suivi de INC A a le même effet que NEG sans influencer C_i. Les autres indicateurs changent avec INC A.

Exemple 4 : Ecrivons un programme pour montrer l'action de XOR pour rendre un texte inintelligible.
Entrez le programme à partir de 43010.

```
43010  11,00,08          LD DE,800h
43013  0E,08             LD C,8
43015  21,00,C0          LD HL,C000
43018  22,F8,A7      ┌──►LD (43000),HL
43021  06,50         ↑   LD B,80d
```

```
43023   7E              →LD A,(HL)
43024   EE,18            XOR 24d
43026   77               LD (HL),A
43027   23               INC HL
43028   10,F9            DJNZ -7+2
43030   2A,F8,A7         LD HL,(43000)
43033   19               ADD HL,DE
43034   0D               DEC C
43035   20,ED            JR NZ,-19+2
43037   C9               RET
```

Entrez un petit BASIC :

```
5 MODE 1
10 PRINT "Ce texte sera transforme par X
OR."
20 END
```

Faites RUN puis CALL 43010 et voyez le texte modifié. Il faut la clé (l'argument de XOR) pour rétablir le texte. Faites CALL 43010 pour le rétablir.

Ce programme emploie les mêmes techniques que pour les exemples précédents. La seule différence de manipulation est l'addition de 8 au registre H pour pointer sur la ligne suivante. On fait ADD HL, DE avec DE contenant 800h.

RÉPONSES AUX EXERCICES

Exercice 1 :



```
43000   3E,20            LD A,32
43002   06,E0            LD B,224
43004   CD,5A,BB        →CALL BB5A
43007   3C               INC A
43008   10,FA            DJNZ -6+2
43010   C9               RET
```

Faites CLS:CALL 43000
Les deux premières instructions peuvent être interverties.

Exercice 2 : Changez les trois lignes suivantes de l'exemple 3.

```
43012  11,4F,C0          LD DE,C04F
  ⋮
43024  2B                DEC HL
  ⋮
43029  ED,B8             LDDR
```

C'est très bien si vous avez trouvé.

Si vous êtes cartésien, changez également la ligne 10 du BASIC en remplaçant le mot gauche par le mot droite.

COMPARAISONS, ROTATIONS, ÉCRAN (SUITE) 4

Toutes les instructions qui modifient un ou plusieurs indicateurs peuvent être suivies par des instructions de sauts conditionnels. Nous examinons, dans cette partie, les instructions de comparaison qui influencent les indicateurs et permettent des décisions suivant le résultat de la comparaison, nous ouvrant ainsi de plus larges possibilités de traitement.

Les rotations et décalages que nous décrivons ensuite, permettent également des orientations diverses selon le but poursuivi. Ces instructions sont très utiles car elles agissent sur un opérande, sur un octet ; les 7 registres simples et les octets mémoire sont concernés. ceci évite le passage obligatoire par A, comme c'est le cas pour toutes les instructions d'un octet qui agissent sur deux opérandes.

Tout d'abord, continuons la description de la mémoire d'écran pour introduire les attributs affectés aux caractères. L'attribut le plus important est la couleur.

1

Mémoire d'écran et la couleur

Nous avons vu à la partie précédente qu'une ligne de pixels se compose de 80 octets. Chaque octet contient 8 bits. Il y a donc 640 pixels par ligne. Dans chaque octet qui constitue la mémoire d'écran, le bit 7 représente, ou se trouve dans le groupe qui représente, le point le plus à gauche et le bit 0 le point le plus à droite.

Selon le mode employé, les points graphiques (appelés aussi points d'affichage ou, tout simplement, points) sont constitués de plusieurs pixels.

EN MODE 2

Il y a autant de points que de pixels. En numérotant les points à partir de la gauche, nous avons pour un octet d'écran :

0	1	2	3	4	5	6	7	points
7	6	5	4	3	2	1	0	bits de l'octet

octet écran

Nous ne disposons que de 2 couleurs : une couleur pour la plume ou le stylo (pour les bits qui sont à 1) et une couleur papier (pour les bits qui sont à 0). Les couleurs par défaut sont jaune vif pour la plume, et bleue pour le papier.

EN MODE 1

Nous avons 4 points pour 8 pixels. L'arrangement est le suivant :

0	1	2	3	points
3 7	2 6	1 5	0 4	bits de l'octet

octet écran

Nous disposons de 2 bits par point : il y a donc 4 couleurs possibles. Les couleurs par défaut sont bleu, jaune vif, turquoise vif et rouge vif. Toutes les fois qu'on a mis, jusqu'ici, 255 dans les octets d'écran, en mode 1, nous avons obtenu du rouge vif. Parmi les combinaisons de deux bits 00, 01,10,11, la dernière est du rouge vif. Ici aussi, le papier est bleu et la plume jaune vif par défaut.

EN MODE 0

Nous avons deux points pour 8 pixels avec l'arrangement suivant :

0	1	points
1 5 3 7	0 4 2 6	bits

octet écran

Nous disposons de 4 bits par point : il y a 16 couleurs possibles. Ici aussi le papier est bleu et la plume jaune vif.
Les couleurs par défaut, ainsi que celles des modes 2 et 1 sont présentées page E3.4 de votre guide de l'utilisateur. En fait, chaque couleur correspond à un numéro d'encre et les seize premières affectations sont faites au cours de l'initialisation. Vous pouvez, bien sûr, affecter d'autres couleurs aux numéros d'encre, mais en mode 2, vous n'avez que deux couleurs et quatre en mode 1. C'est pour cette raison que la commande BASIC MODE x effectue un CLS et une remise de nombreux paramètres du système. On ne peut employer plusieurs modes sur le même écran.

2

Les comparaisons et tests

LES TESTS D'UN BIT

CB,47 BIT 0,A LET Z=1−Ao

réalise l'opération : $Z=R_b$ qui agit sur l'indicateur Z. C'est le complément d'un bit d'un des sept registres simples. Voyez les 56 instructions possibles en annexe.

Pour tester un bit d'un octet mémoire, nous avons :

CB,46 BIT 0,(HL)

qui effectue : $Z=(HL)_b$.
Voyez, en annexe, les 24 instructions possibles sur des octets dont l'adresse est tenue par HL, IX ou IY.

BIT donne le moyen de tester n'importe quel bit (0 à 7) de n'importe quel registre ou octet mémoire. Les sauts conditionnels Z ou NZ s'emploient ensuite pour orienter l'instruction suivante.

Les codes des caractères ASCII affichables vont de 20 à 7E (32 à 126d) ; voir page A3.1 du guide de l'utilisateur. Des tests sur des bits du quartet haut permettent leur identification. Par exemple, les lettres majuscules vont de 41h à 5A et les minuscules de 61h à 7A. Les

minuscules ont les mêmes codes que les majuscules mais avec le bit 5 à 1. Elles ont la valeur du code majuscule plus 32d, 20h. Le test du bit 5 renseigne sur l'état de la lettre.

Le test du bit 0 renseigne si l'octet est pair ou impair.
Le test du bit 7 est souvent inutile, car le bit 7 est reflété dans l'indicateur S. Encore faut-il que la dernière instruction ait influencé cet indicateur. Par exemple LD A,(DE) n'influence aucun indicateur, et l'état de S dépend des instructions précédentes. Dans ce cas, BIT 7,A est nécessaire pour savoir si A est plus grand que 127.

LES COMPARAISONS

```
        FE,nn                   CP n
exécute : A−n
        B8                      CP B
exécute : A−B
        BE                      CP (HL)
exécute : A− (HL)
```

Les comparaisons d'une constante, d'un registre ou d'un octet mémoire ne se font qu'avec A.
Les mnémoniques CP effectuent une soustraction sans modifier d'octets : il n'y a pas de résultat. Les indicateurs sont influencés par cette instruction comme si la soustraction avait lieu. Ces instructions sont très souvent employées, suivies de sauts conditionnels, d'autant plus qu'elles ne modifient que les indicateurs.
La correspondance BASIC pourrait être :

IF comparaison THEN saut ou décision

Par exemple :

CP 32d

fait :

IF A>32 THEN C_i=0
IF A<32 THEN C_i=1
IF A=32 THEN Z=1
etc.

Le test sur C_i indique si A contient un code de caractère affichable. Le test sur Z indique si A contient le code “espace”.
Les C.M. ci-dessous permettent de savoir si C=0 :

```
        LD A,0
        CP C                    ;IF C=0
        JR Z,e                  ;THEN GOTO
```

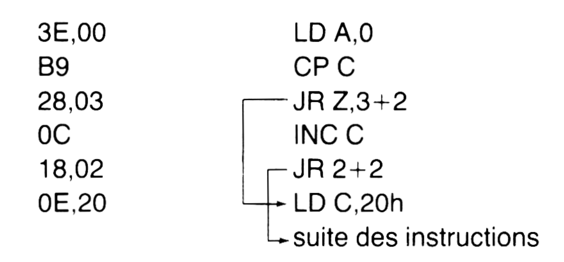
ou bien

```
LD A,C
CP 0
JR Z,e
```

Le THEN sera toujours un THEN GOTO mais, à l'adresse de transfert, vous mettez des C.M. équivalents à PRINT, LET, etc., selon le but recherché. Par exemple, si vous voulez traduire :

IF C=0 THEN C=32 ELSE C=C+1

faites :



Exemple 1 : Inversons tous les bits d'un fichier. Pour ce faire, prenons le fichier d'écran.

```
43000  21.00.C0        LD HL,C000      ;debut ecran
43003  7E              LD A,(HL)       :saisie un octet
43004  2F              CPL             ;l'inverse
43005  77              LD (HL),A       :le reaffiche
43006  23              INC HL          ;octet suivant
43007  7C              LD A,H
43008  FE,FF           CP FF           :H=FF?
43010  20,F7           JR NZ,-9+2      :si non.saut
43012  7D              LD A,L          :si oui
43013  FE,FF           CP FF           ;L=FF?
43015  20,F2           JR NZ,-14+2     :si non, saut
43017  C9              RET             :si oui
```

Rien de spécial dans ce programme, mis à part le fait qu'il montre l'emploi de CP pour savoir si HL a atteint FFFF.

Faites : LIST
puis : CALL 43000
Faites plusieurs fois CALL 43000

Vous attendiez-vous à ce résultat ?

Faites MODE 2 puis LIST puis CALL 43000. Voyez-vous la petite barre noire dans la ligne 10220 ? Quand HL a atteint FFFF, on sort du programme ; or cette adresse doit aussi être inversée. Au lieu de faire CP FF, faisons CP 00 en entrant la commande POKE 43009,0. De plus, quand H=0, L est aussi à 0, on n'a plus besoin de la comparaison de L. Entrons la commande POKE 43012,&C9. Faites MODE 2 puis LIST puis CALL 43000. La petite barre a disparu.

Exercice 1 : Il y a un autre moyen pour contrôler la fin du fichier d'affichage, vu avec les instructions logiques. Modifiez le programme de l'exemple 1. Au lieu des 6 lignes 43008 à 43017, écrivez-en 3.

Exemple 2 : Reprenons l'affichage d'un texte sur l'écran. Dans un premier exemple, nous employions une succession d'ordre PRINT en appelant la routine BB5A plusieurs fois (exercice 1 chapitre 2). Dans un deuxième exemple, nous employions une boucle pour n'écrire qu'une fois CALL BB5A (ex. 1 partie 3). Dans cet exemple-ci, nous employons la comparaison avec un indicateur de fin de message.

Signalons que la routine BB5A accepte également les codes de contrôle - de 0 à 31d - et qu'elle les exécute. Ce que les codes de contrôle font exécuter est repris au chapitre 9 du guide de l'utilisateur. Quand un code de contrôle demande un ou plusieurs paramètres (jusqu'à 9 pour le code 25d), il *faut* les envoyer tous.

Ecrivons un programme qui affiche un texte :

```
43000  0C,1F,04,05            DEF codes de controle: cls; puis
                                              :colonne 4,ligne 5
43004  4D,6F,6E,20            DEF Mon_
43008  6F,72,64,69,6E,61      DEF ordina
43014  74,65,75,72,20         DEF teur_
43019  65,73,74,20            DEF est_

43023  6D,65,72,76            DEF merv
43027  65,69,6C,6C            DEF eill
43031  65,75,78,2E            DEF eux.
43035  07                     DEF code de controle: emet un son
43036  FF                     DEF fin de message

43040  21,F8,A7               LD HL,43000      ;debut fichier
43043  7E                  ┌─►LD A,(HL)        ;charge A
43044  FE,FF               │  CP FF            ;fin de message?
43046  28,06             ┌─┼──JR Z,6+2         ;si oui, retour
43048  CD,5A,BB          │ │  CALL BB5A        ;si non, affiche
43051  23                │ │  INC HL           ;caractere suivant
43052  18,F5             │ └──JR -11+2         ;saut a 43043
43054  C9                └───►RET
```

Faites CALL 43040.

Nous avons choisi FF comme indicateur de fin de message. Dès que A contient FF, il y a saut à RET.
Le code de contrôle OC exécute un CLS.
Les codes de contrôle 1F,04,05 positionnent le curseur à la quatrième colonne et cinquième ligne.
Le code de contrôle 07 émet un son.
Le reste va de soi.

Exemple 3 : Ecrivons un programme pour échanger deux fichiers. Les applications en sont multiples.
Gardons le programme précédent en mémoire. Entrons nos codes à partir de 43060.

```
43060  21,F8,A7           LD HL,43000      ;fichier 1
43063  11,5C,A8           LD DE,43100      ;fichier 2
43066  1A              ┌─►LD A,(DE)        ;fichier 2 dans A
43067  46              │  LD B,(HL)        ;fichier 1 dans B
43068  77              │  LD (HL),A        ;A dans fichier 1
43069  78              │  LD A,B           ;A=B
43070  12              │  LD (DE),A        ;B dans fichier 2
43071  13              │  INC DE           ;octet suivant
43072  23              │  INC HL           ;  "       "
                       │
43073  7D              │  LD A,L           ;A=L
43074  FE,1F           │  CP 1F            ;fini?
43076  20,F4           └──JR NZ,-12+2      ;non, saut
43078  C9                 RET              ;oui, retour
```

Rien de spécial dans ce programme, sauf pour contrôler la sortie. Nous échangeons le fichier de 40 octets, qui va de 43000 à 43039 (A7F8 à A81F), avec un autre qui débute à 43100. On teste à quel moment L arrive à 1F pour prendre une décision.

Entrons un fichier à l'adresse 43100 :

```
43100  0C,1F,06,0F,18       DEF codes de controle
43105  51,75,27,65,6E,20    DEF Qu'en_
43111  70,65,6E,73,65,20    DEF pense_
43117  76,6F,74,72,65,20    DEF votre_
43123  63,6F,6E,6A,6F       DEF conjo
43128  69,6E,74,3F          DEF int?
43132  07                   DEF code de controle
43133  FF                   DEF fin de message
```

Faites CALL 43040, puis faites CALL 43060:CALL 43040.

A l'adresse 43104, nous avons ajouté le code de contrôle 24d qui inverse les couleurs plume et papier. Pour voir et comprendre son influence, faites plusieurs fois CALL 43040. Ensuite plusieurs fois CALL 43060:CALL 43040.

LES COMPARAISONS AVEC RÉPÉTITION

```
ED,A9          CPD
```

réalise plusieurs opérations :

```
A-(HL)
HL=HL-1
BC=BC-1
```

qu'on peut traduire en BASIC :

```
IF A=PEEK(HL) THEN ...
```

```
ED,A1          CPI
```

qui réalise :

```
A-(HL)
HL=HL+1
BC=BC-1
```

```
ED,B9          CPDR
ED,B1          CPIR
```

qui réalisent les mêmes opérations que CPD et CPI mais avec répétition jusqu'à ce que A=(HL) ou que BC=0.

Ces instructions agissent sur l'indicateur S de la manière habituelle. Elles n'ont pas d'influence sur C_i. Elles agissent sur l'indicateur Z comme suit :

Z=1 si A=(HL)

et sur l'indicateur V comme suit :

V=0 si BC=0 après exécution, si non V=1

Après l'une de ces instructions, le programme se poursuit tout simplement, ou un saut conditionnel est effectué selon le but poursuivi.
Ces mnémoniques sont très puissantes. Elles servent, entre autres, à retrouver la première occurrence d'un caractère dans un fichier, ou la première occurrence d'une chaîne.
Les instructions CPDR et CPIR permettent plusieurs milliers d'instructions avec une seule mnémonique.

Exemple 4 : Ecrivons un programme pour protéger votre programme BASIC avec une clé. Cette clé se compose de 3 lettres.

```
43000   CD.06,BB           CALL BB06         ;attend une touche
43003   FE.58              CP 58             ;code de X
43005   20.0F            ┌─JR NZ.15+2        ;si faux:RAZ
43007   CD.06.BB         │ CALL BB06
43010   FE.41            │ CP 41             ;code de A
43012   20.08            ●─JR NZ.8+2         ;si faux:RAZ
43014   CD.06,BB         │ CALL BB06
43017   FE.7A            │ CP 7A             ;code de z
43019   20.01            ●─JR NZ.1+2         ;si faux:RAZ
43021   C9               │ RET
43022   CD.00,00         └→CALL 0            ;RAZ
```

Nous employons une routine du système, d'adresse BB06. Cette routine scrute le clavier et attend qu'une touche soit pressée. Dès que cela se produit, la routine retourne le code ASCII de la touche pressée dans A. Pas de condition d'entrée ; en sortie, le registre F a changé.
On appelle la routine 0 pour effectuer une RAZ (Remise A Zéro) générale. Elle correspond aux touches CTRL-SHIFT-ESC pressées en même temps.
La clé employée se compose des 3 lettres – X A z (deux majuscules et une minuscule) – à entrer successivement dans cet ordre. Par exemple, entrez le BASIC suivant :

```
10 CLS:PRINT "Entrez la cle du programme
"
20 CALL 43000
30 PRINT "Je possede 240.000.- frs en Su
isse"
40 REM etc...
9999 END
```

Quand le programme tourne, vous devez entrer X puis A puis z et le programme poursuit. Si vous vous trompez, tout s'efface et le message de mise sous tension apparaît.

Exercice 2 : Nous vous proposons un exercice difficile. Ecrivez un programme semblable à celui de l'exemple 4. Employez une boucle comprenant CALL BB06 suivi de CPI. Mettez les 3 codes dans un tampon à 43000. Votre programme commence à 43003. Faites attention aux sauts conditionnels.

3

Les rotations et glissements

LES ROTATIONS

17 RLA

effectue une rotation de 9 bits à gauche avec l'indicateur C sur A.

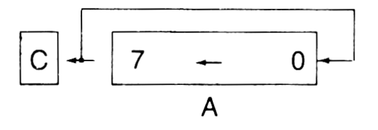
C ← 7 ← 0

A

Le bit 7 de A va dans C_i, C_i va dans le bit 0 de A, le bit 0 dans le bit 1, etc.

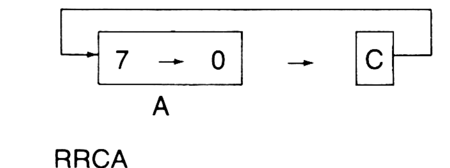
07 RLCA

effectue une rotation de huit bits à gauche sur A. Le bit 7 de A va dans le bit 0 de A et dans C_i.



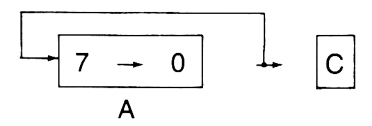
1F RRA

effectue une rotation de neuf bits à droite avec l'indicateur C sur A.



0F RRCA

effectue une rotation de huit bits à droite sur A. Le bit 0 va dans le bit 7 et dans C_i.



Ces quatre instructions, qui agissent sur A, n'influencent que C_i. Par contre, les rotations, sur neuf ou huit bits, à gauche ou à droite, d'un registre simple ou d'un octet mémoire (voir les C.M. en annexe) agissent sur les indicateurs S, Z, V et C.

17	RLA
CB,17	RL A

font la même rotation. C'est la même chose sauf pour les indicateurs.

Remarque : Notez que l'espace, dans la mnémonique, a de l'importance. RLA n'est pas RL A ; CPL n'est pas CP L ; etc.

LES GLISSEMENTS

CB,3F SRL A

0 → 7 → 0 → C
A

effectue un glissement logique à droite. 0 va dans le bit 7, le bit 7 dans le bit 6, ..., le bit 0 dans C_i.

CB,2F SRA A

7 → 0 → C
A

effectue un glissement arithmétique à droite. Le bit 7 ne change pas et va dans le bit 6, ..., le bit 0 va dans C_i.

CB,27 SLA A

C	←	7 ← 0	←0
		A	

effectue un glissement arithmétique à gauche. 0 va dans le bit 0, ..., le bit 7 dans C_i.
Voyez les C.M. des glissements d'un registre ou d'un octet mémoire en annexe. Tous les glissements agissent sur les 4 indicateurs principaux.

Après un SRA, le bit 6 sera toujours le même que le bit 7. L'effet de SRA est de diviser par 2 les nombres positifs et négatifs (avec signe).

1	1	1	1	1	1	0	0	FC devient, après SRA
1	1	1	1	1	1	1	0	FE

donc −4 devient −2.

SRA appliqué à un nombre impair, rend un résultat faux. Un test sur C_i le signale.
SRL divise par 2 les nombres positifs (binaire pur), avec la même restriction pour les nombres impairs.
SLA multiplie par 2. L'indicateur C signale s'il y a mod. 256.

0	0	0	0	1	0	1	0	A devient après SLA
0	0	0	1	0	1	0	0	14

donc 10 devient 20.

Pour multiplier par 2 un nombre de 16 bits, on fait :

```
SLA (pq)
RL (pq+1)
```

suivi d'un test sur C_i si on veut savoir si le résultat est modulo 65536.
Pour multiplier par 2 un nombre de 24 bits, on fait :

```
SLA (pq)
RL (pq+1)
RL (pq+2)
```

Le test sur C_i renseigne modulo 16777216.
Pour multiplier par 3, on fait un glissement, suivi d'une addition.
Les rotations et glissements agissent au niveau du bit, ce qui correspond au pixel sur l'écran.

Exemple 5 : Faisons défiler l'écran, vers la gauche, d'un pixel à la fois.
A l'aide du programme de chargement, commençons à 43010. Nous ne ferons plus cette remarque. Vous savez introduire des C.M. à l'adresse voulue ; il vous suffit de lire la première adresse de nos programmes.

```
43010  21,4F,C0     LD HL,C04F       ;fin 1iere ligne
43013  16,02        LD D,2           ;2 rangees
43015  0E,08        LD C,8           ;8 lignes/rangee
43017  22,F8,A7     LD (43000),HL    ;adresse rangee
43020  06,50        LD B,80          ;80 octets/ligne
43022  22,FA,A7     LD (43002),HL    ;adresse ligne
43025  7E           LD A,(HL)        ;saisie
43026  17           RLA              ;rotation a gauche
43027  77           LD (HL),A        ;stockage
43028  2B           DEC HL           ;suivant
43029  10,FA        DJNZ -6+2
43031  2A,FA,A7     LD HL,(43002)    ;recupere
43034  CB,13        RL E             ;Ci dans Eo
43036  CB,3E        SRL (HL)         ;elimine (HL)o
43038  CB,1B        RR E             ;Eo dans Ci
43040  CB,16        RL (HL)          ;Ci dans (HL)o
43042  3E,08        LD A,8           ;ajoute 800h
43044  84           ADD A,H          ;  a HL
43045  67           LD H,A           ;
43046  0D           DEC C
43047  20,E3        JR NZ,-29+2
43049  2A,F8,A7     LD HL,(43000)    ;recupere
43052  7D           LD A,L           ;ajoute
43053  C6,50        ADD A,80d        ;  50h
43055  6F           LD L,A           ;  a
43056  7C           LD A,H           ;  HL
43057  CE,00        ADC A,0          ;
43059  67           LD H,A           ;
43060  15           DEC D
43061  20,D0        JR NZ,-48+2
43063  C9           RET
```

On part de la fin de la première ligne, puis on fait des rotations à gauche sur chaque octet. Une rotation met le bit 7 de (HL) dans C_i et la suivante met C_i dans le bit 0 de (HL−1). La boucle interne effectue ces rotations 80 fois pour les 80 octets d'une ligne.

Pour avoir une rotation complète, nous devons reprendre le bit 7 du dernier octet ayant subi la rotation (C000) pour le mettre dans le bit 0 du premier octet (C04F). Ce transfert s'effectue en mettant C_i provisoirement dans Eo afin de le réintroduire dans (C04F)o, aux lignes 43034 à 43040.

Ensuite, on ajoute 800h à HL pour effectuer la ligne suivante de la première rangée de caractères. Les 8 lignes sont contrôlées par la deuxième boucle.

Ensuite, on ajoute 50h à l'adresse de la première ligne de rangée de caractères précédente. C'est l'objet de la troisième boucle. Il suffit de changer l'octet d'adresse 43014 pour avoir plus de rangées à défiler.

Entrez le BASIC suivant :

```
10 MODE 1
20 PRINT "Voici un texte qui montre la r
otation"
30 PRINT "de deux rangees de caracteres
a gauche"
40 FOR n=1 TO 640
50 CALL 43010
60 NEXT
70 END
```

Faites RUN.
En mode 1, chaque point est représenté par deux pixels (pour les couleurs), d'où changement de couleur.
Faites 10 MODE 2 puis RUN.
En mode 2, chaque point est représenté par un pixel, d'où la continuité du défilement.

Pour corriger cela, les instructions 43025 à 43028 devraient être répétées deux fois en mode 1 et quatre fois en mode 0, et en modifiant également la valeur poussée dans B. Nous verrons d'autres moyens par la suite.

Exercice 3 : Ecrivez un programme pour défiler à droite d'un pixel à la fois. C'est le même programme que l'exemple précédent dont vous changez cinq instructions.

4

Les échanges et la pile

LES ÉCHANGES

Nous avons vu, dans la partie 1 figure 2, que le microprocesseur possède un deuxième jeu de registres. Voici les C.M. qui permettent d'y accéder.

08 EX AF,A'F'

réalise deux échanges : A <=> A' et F <=> F'.
Tous les indicateurs d'état sont modifiés puisque F change.
La traduction BASIC peut être :

```
let X=A
let A=A'
let A'=X
```

Idem pour F

D9 EXX

réalise les échanges des trois registres couplés, donc des six registres simples restants :

BC <=> B'C' , DE <=> D'E' , HL <=> H'L'

Dans l'Amstrad, on ne doit pas employer le deuxième jeu de registres parce que le système opératoire emploie ce deuxième jeu pour y stocker des variables et des indicateurs. Nous avons signalé ces deux instructions pour être complet.

```
EB              EX DE,HL
```

réalise l'échange : DE <=> HL.

```
E3              EX (SP),HL
DD,E3           EX (SP),IX
FD,E3           EX (SP),IY
```

réalisent deux échanges : (SP) <=> RRbas et (SP+1) <=> RRhaut.
Ces quatre instructions n'ont pas d'effet sur les indicateurs.
Pour faire ADD DE,BC qui n'existent pas, on emploie :

```
EB              EX DE,HL
09              ADD HL,BC
EB              EX DE,HL
```

et HL ne change pas.
Pour connaître une adresse de retour, située dans la pile, l'instruction E3 est intéressante :

```
E3              EX (SP),HL
22,mm,nn        LD (nm),HL
E3              EX (SP),HL
```

Aucun registre ne change, et l'adresse de retour de la routine est stockée à l'adresse nm.

LA PILE

L'espace mémoire, réservé pour la pile au cours de l'initialisation, est situé juste en dessous de C000.
La pile s'utilise très souvent pour stocker provisoirement le contenu des différents registres. Son emploi requiert quelques précautions.

```
C5              PUSH BC
```

exécute plusieurs opérations :

```
(SP-1)   =BChaut
(SP-2)   =BCbas
SP       =SP-2
```

C'est un moyen de stocker BC dans la pile. Pour le récupérer, on fait :

```
C1          POP BC
```

exécute plusieurs opérations :

```
BCbas    =(SP)
BChaut   =(SP+1)
SP       =SP+2
```

(Voir en annexe les autres C.M. pour les registres doubles et couplés.)

PUSH et POP n'influencent pas les indicateurs. Ces deux instructions sont très employées. Outre le stockage provisoire de registres, elles servent à faire des échanges, à modifier une adresse de retour, etc.

Attention : Compensez toujours un PUSH par un POP avant le RET de votre routine. Le système Amstrad et l'interpréteur BASIC utilisent également cette pile.

L'utilisateur de la pile permet de pallier l'absence de certaines mnémoniques pour effectuer des opérations très courantes. Nous avons déjà employé certains moyens pour les réaliser, mais l'emploi de la pile est plus rationnel.

Pour faire LD BC,HL :

```
PUSH HL
POP BC
```

Pour faire ADD BC,DE :

```
PUSH BC
POP HL
ADD HL,DE
PUSH HL
POP BC
```

Pour faire ADD B,n :

```
PUSH AF
LD A,B
ADD A,n
LD B,A
POP AF
```

Pour faire EX D,E :

```
PUSH AF
LD A,D
LD D,E
LD E,A
POP AF
```

Pour faire LD C,F :

```
PUSH AF
POP BC
```

RÉPONSES AUX EXERCICES

Exercice 1 :

```
43008  B5                 OR L
43009  20,F8              JR NZ,-8+2
43011  C9                 RET
```

Si vous n'avez pas oublié de changer l'argument de JR NZ, c'est très bien.

Exercice 2 :

```
43000  58,41,7A           DEF XAz
43003  21,F8,A7           LD HL,43000
43006  01,03,00           LD BC,3
43009  CD,06,BB           CALL BB06
43012  ED,A1              CPI
43014  E2,0E,A8           JP PO,43022     :RET si V=0
43017  28,F6              JR Z,-10+2      :saut si A=(HL)
43019  CD,00,00           CALL 0
43022  C9                 RET
```

La ligne 20 du BASIC de l'exemple 4 devient CALL 43003. C'est très très bien si vous avez réussi ce programme ; vous avez une bonne compréhension de CPI.

Exercice 3 : Les 5 lignes qui changent sont les suivantes :

```
43010  21,00,C0        LD HL,C000     ;debut 1iere ligne
43026  1F              RRA            ;rotation a droite
43028  23              INC HL
43036  CB,26           SLA (HL)       ;elimine (HL)7
43040  CB,1E           RR (HL)        ;C dans (HL)7
```

Même programme d'exploitation ou un autre de votre choix.

QUELQUES PROGRAMMES UTILES | 5

Nous terminons la revue des instructions du microprocesseur en présentant les C.M. qui appellent et qui terminent les routines, ainsi que les instructions d'entrée/sortie. Celles-ci sont surtout employées par le système Asmtrad. Il est bon de les connaître. Elles sont nécessaires pour ceux qui réaliseront des circuits enfichables, et utiles pour ceux qui désirent explorer le système d'exploitation. Ce système, dont nous employons encore quelques routines, est examiné dans la partie suivante.

Nous décrivons également quelques programmes d'utilité générale, tels que la génération de nombre aléatoire, les retards programmés, la scrutation du clavier, etc. Tous ces programmes seront utilisés pour la réalistion d'un jeu d'adresse en fin de partie.

1

Les appels et les retours de routines

Dans la seconde partie, nous avons employé la mnémonique CALL pour appeler une routine, en écrivant son adresse comme paramètre. Voici le détail de cette instruction :

CD,qq,pp	CALL pq	GOSUB pq

qui exécute les opérations :

(SP−1)	=PC haut
(SP−2)	=PCbas
SP	=SP−2
PC	=pq

Dans un programme en C.M., CALL pq provoque un saut à une routine d'adresse pq. L'adresse suivante du programme, actualisée par l'instruction, est sauvée dans la pile. La routine appelée se termine par un RET qui provoque le retour dans le programme qui a appelé la routine.

LES APPELS CONDITIONNELS

C4,qq,pp	CALL NZ,pq	si Z=0
CC,qq,pp	CALL Z,pq	si Z=1
D4,qq,pp	CALL NC,pq	si C_i=0

DC,qq,pp	CALL C,pq	si $C_i=1$
E4,qq,pp	CALL PO,pq	si P/V=0
EC,qq,pp	CALL PE,pq	si P/V=1
F4,qq,pp	CALL P,pq	si S=0
FC,qq,pp	CALL M,pq	si S=1

Si la condition est réalisée, alors il y a appel, et :

(SP−1) =PChaut
(SP−2) =PCbas
SP =SP−2
PC =pq

Si la condition est fausse, alors :

PC=PC+3 et le programme continue.

Les routines se terminent par :

C9	RET	RETURN

qui effectue :

PCbas =(SP)
PChaut =(SP+1)
SP =SP+2

et ramène le déroulement du programme à l'endroit où l'appel de la routine a été programmé.

LES RETOURS CONDITIONNELS

C0	RET NZ	si Z=0
C8	RET Z	si Z=1
D0	RET NC	si C=0
D8	RET C	si C=1
E0	RET PO	si P/V=0
E8	RET PE	si P/V=1
F0	RET P	si S=0
F8	RET M	si S=1

Si la condition est réalisée, alors :

PCbas =(SP)
PChaut =(SP+1)
SP =SP+2

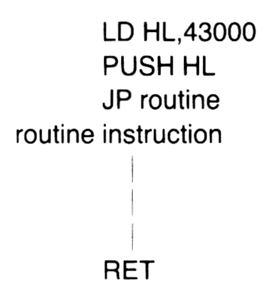
Les CALL et RET n'ont aucune influence sur les indicateurs.
La routine appelée par CALL, peut contenir des POP et PUSH avant le RET pour altérer l'adresse de retour :

```
POP HL
INC HL
PUSH HL
RET
```

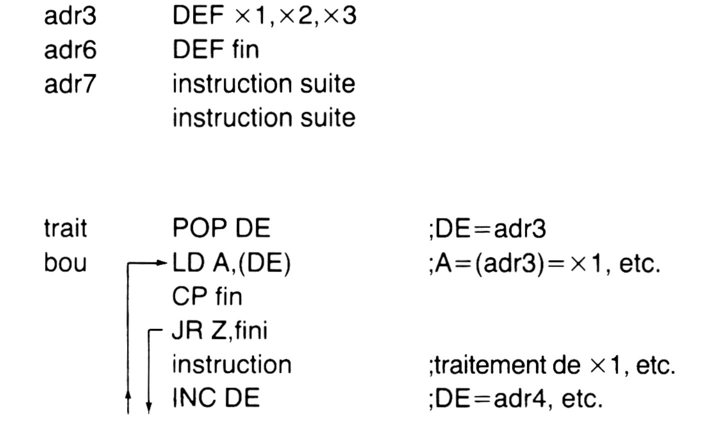
qui augmente l'adresse de retour de 1 :

```
POP HL
LD HL,43200
PUSH HL
RET
```

qui change l'adresse de retour en 43200.
Il est possible également de faire un saut dans une routine en employant JP ou JR ; il faut alors avoir poussé l'adresse de retour dans la pile.



et le retour se fait à 43000.
On peut également pousser des données, constantes ou variables, après un CALL, et les exploiter dans la routine.



```
              JR bou
fini          INC DE          ;DE=adr7
              PUSH DE         ;adr7 dans la pile
              RET
```

L'emploi des C.M. permet ainsi des raccourcis de programmation en manipulant les adresses de retour. Ceci montre également toute la puissance potentielle de la pile. En connaissant parfaitement son mécanisme, vous l'emploierez avec profit.

LES REDÉMARRAGES

C7	RST 0	q=0
CF	RST 8	q=8
D7	RST 10	q=16d
DF	RST 18	q=24d
E7	RST 20	q=32d
EF	RST 28	q=40d
F7	RST 30	q=48d
FF	RST 38	q=56d

exécute les opérations :

(SP−1)	=PChaut
(SP−2)	=PCbas
SP	=SP−2
PChaut	=0
PCbas	=q

Un RST travaille comme un CALL, mais ne nécessite qu'un C.M. au lieu de trois. Chaque RST comporte 8 octets sauf le dernier −RST 38− qui commence à 56d et qui peut s'étendre sur plus de 8 octets.

Les RST sont employés par les concepteurs de système pour différents usages. Le système Amstrad en fait un usage qui lui est propre, décrit dans le chapitre suivant. Ne les employez pas, sauf le premier : RST 0, qui effectue une RAZ générale. Nous l'avons déjà employé en faisant CALL 0.

2

Entrées et sorties de périphériques

De même que les octets mémoire ont une adresse, au moyen de laquelle on manipule ces octets, de même les périphériques ont une adresse au moyen de laquelle on communique avec eux. Les adresses possibles vont de 0 à 65535. Quand il s'agit d'une adresse de périphérique, on l'appelle également un port.

Un périphérique est un matériel externe au système de base. Tout système de base comporte le microprocesseur, la mémoire RAM et ROM – MEV et MEM –, et un circuit spécialisé ULA. Suivant la figure 1 de la première partie, vous voyez que l'Amstrad possède des périphériques incorporés dans son boîtier, et qu'il permet d'autres périphériques grâce aux différents connecteurs.

Les mnémoniques LD servent à échanger des informations entre le microprocesseur et la mémoire ; de même, les mnémoniques IN et OUT réalisent les échanges entre le microprocesseur et les périphériques.

DB,pp IN A,(p) A=PEEK(p)

exécute l'opération : A=(p).
Le contenu de l'adresse du port p est chargé dans A.

D3,pp OUT (p),A

qui exécute : (p)=A.

Le contenu de A est chargé dans l'adresse du port p.
L'adresse du port est Ap où A est l'octet haut et p l'octet bas.
Ces 2 C.M. n'ont aucune influence sur les indicateurs.

Les échanges se réalisent également avec les autres registres simples.

```
ED,40          IN B,(C)          B=PEEK(BC)
ED,41          OUT (C),B         POKE BC,B
```

Voyez les autres C.M. en annexe. BC donne l'adresse du port dont le contenu entre dans un registre simple R, ou est chargé par ce registre R.

Les instructions OUT (C),R n'ont pas d'influence sur les indicateurs. Les instructions IN R,(C) agissent sur S, Z et P/V *pas sur* C_i.
Amstrad utilise les instructions dont l'adresse est donnée par BC: B spécifie le périphérique et C la donnée.

Par exemple :

```
01,89,7F       LD BC,7F89
ED,49          OUT (C),C         ;(7F89)=89h
```

le port 7F reçoit la donnée 89h.

Les échanges entre périphériques et octets mémoire se réalisent avec les instructions :

```
ED,A2          INI
```

effectue les opérations :

```
(HL) =(C)
HL   =HL+1
B    =B-1
```

```
ED,AA          IND
```

effectue :

```
(HL) =(C)
HL   =HL-1
B    =B-1
```

```
ED,A3          OUTI
```

qui effectue :

```
(C)  =(HL)
HL   =HL+1
B    =B-1
```

ED,AB OUTD

effectue :

(C) =(HL)
HL =HL−1
B =B−1

Ces quatre instructions influencent Z de la manière suivante : Z=1 si B=0 après exécution, si non Z=0.

ENTRÉES ET SORTIES AVEC RÉPÉTITION

ED,B2 INIR

effectue les mêmes opérations que INI, mais avec répétition jusqu'à ce que B=0.

ED,BA INDR

effectue les mêmes opérations que IND avec répétition jusqu'à ce que B=0.

ED,B3 OTIR
ED,BB OTDR

effectuent les mêmes opérations que OUTI et OUTD mais avec répétition jusqu'à ce que B=0.

Ces quatre instructions mettent Z à 1.

Ces quatre instructions sont aussi puissantes que leurs homologues de chargement et de comparaison avec répétition. Elles sont très utiles pour manipuler les données sur disquettes. La sauvegarde des programmes, la gestion de fichiers, les échanges de données, etc., se font à grande vitesse.

3

Quelques programmes

AFFICHAGE EN DOUBLE HAUTEUR

Proposons-nous d'afficher un texte en double hauteur. Pour commencer, doublons la hauteur du texte affiché sur la première rangée ; la deuxième rangée devra rester vierge ou sera suraffichée. Employons LDIR pour transférer les 80 octets d'une ligne dans une autre ligne d'écran. Avant chaque LDIR, nous devons mettre 80 dans BC, et manipuler HL et DE pour que ces deux registres couplés pointent sur la ligne adéquate. Reprenons les adresses des deux premières rangées données dans la troisième partie, figure 1.

			Début de ligne
Première rangée	ligne	1	C000
	ligne	2	C800
	ligne	3	D000
	ligne	4	D800
	ligne	5	E000
	ligne	6	E800
	ligne	7	F000
	ligne	8	F800

Deuxième rangée	ligne	9	C050
	ligne	10	C850
	ligne	11	D050
	ligne	12	D850
	ligne	13	E050
	ligne	14	E850
	ligne	15	F050
	ligne	16	F850

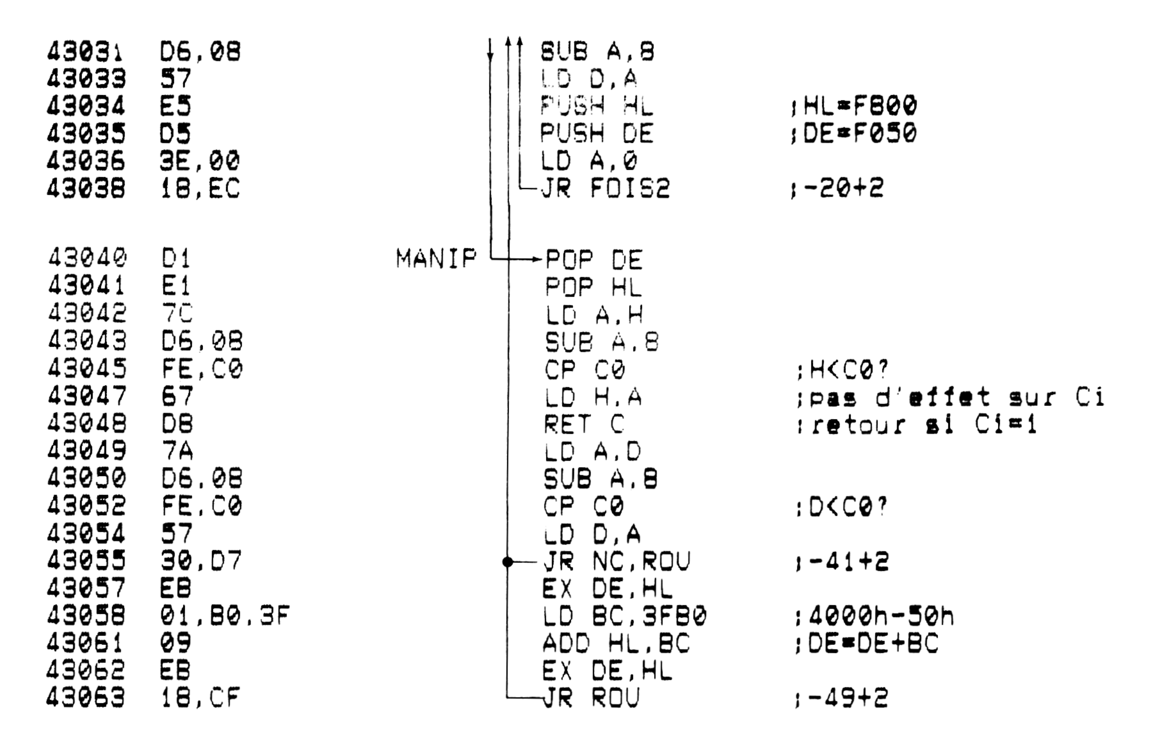
Nous devons transférer la ligne 8 dans la ligne 16, puis dans la ligne 15 ; ensuite la ligne 7 dans la ligne 14, puis 13 ; etc. ; enfin la ligne 1 dans la ligne 2, puis 1. Nous aurons ainsi la double hauteur de ce qui est affiché sur la première rangée.

Nous devons changer successivement les valeurs de HL et DE de la manière suivante :

HL	DE
F800	F850
F800	F050
F000	E850
F000	E050
E800	D850
E800	D050
E000	C850
E000	C050
D800	F800
D800	F000
D000	E800
D000	E000
C800	D800
C800	D000
C000	C800
C000	C000

Voici le programme :

```
43010  11,50,F8                LD DE,F850      ;ligne 16
43013  21,00,F8                LD HL,F800      ;ligne 8
43016  E5          ROU       ┌→PUSH HL
43017  D5                    │ PUSH DE
43018  3E,01                 │ LD A,1
43020  01,50,00    FOIS2     │┌→LD BC,50h      ;80 octets a
43023  ED,B0                 ││ LDIR           ;  transferer
43025  B7                    ││ OR A           ;A=1 puis =0
43026  28,0C                ┌┼┼─JR Z,MANIP     ;+12+2
43028  D1                   │││ POP DE
43029  E1                   │││ POP HL
43030  7A                   ↓↑↑ LD A,D
```



```
43031  D6,08              SUB A,8
43033  57                 LD D,A
43034  E5                 PUSH HL        ;HL=F800
43035  D5                 PUSH DE        ;DE=F050
43036  3E,00              LD A,0
43038  18,EC              JR FOIS2       ;-20+2

43040  D1          MANIP  POP DE
43041  E1                 POP HL
43042  7C                 LD A,H
43043  D6,08              SUB A,8
43045  FE,C0              CP C0          ;H<C0?
43047  67                 LD H,A         ;pas d'effet sur Ci
43048  D8                 RET C          ;retour si Ci=1
43049  7A                 LD A,D
43050  D6,08              SUB A,8
43052  FE,C0              CP C0          ;D<C0?
43054  57                 LD D,A
43055  30,D7              JR NC,ROU      ;-41+2
43057  EB                 EX DE,HL
43058  01,B0,3F           LD BC,3FB0     ;4000h-50h
43061  09                 ADD HL,BC      ;DE=DE+BC
43062  EB                 EX DE,HL
43063  18,CF              JR ROU         ;-49+2
```

Nous avons mis trois étiquettes à certains endroits de la routine. Aux lignes 43026, 43038, 43055 et 43063, nous avons employé ces étiquettes. Ceci illustre un des avantages d'un programme Assembleur : celui-ci calcule automatiquement les sauts quand il rencontre une étiquette dédiée dans un champ opérande d'une mnémonique. En C.M. nous calculons ces sauts ; le calcul est mis en commentaires.

Pour que LDIR s'effectue deux fois avec la même valeur de HL, nous employons A pour ce contrôle. OR A ne change pas A mais influence l'indicateur Z.

La manipulation de HL et DE consiste à les diminuer de 800h. On commence par HL, quand H est inférieur à C0, il y a retour. En diminuant DE il y a un deuxième problème ; quand DE=C050, sa valeur suivante est F800. On réalise ceci aux quatre lignes 43057 à 43062. On aurait pu mettre LD DE,F800, mais nous désirons étendre l'utilisation de cette routine. Telle quelle, on ne peut influencer que la première rangée, et, l'on doit mettre une déclaration MODE dans le BASIC suivant :

```
10 MODE 1
20 PRINT "DOUBLE hauteur"
30 PRINT:PRINT
40 CALL 43010
50 END
```

Faites RUN.

Vous pouvez intercaler la ligne d'attente :

35 FOR n=0 TO 1000:NEXT

Vous pouvez également modifier la ligne 10, car le programme fonctionne dans les 3 modes. Pour agrandir n'importe quelle ligne, ajoutons un programme :

```
43100  2A,F8,A7       ENT     LD HL,(43000)    ;No de rangee
43103  7D                     LD A,L
43104  FE,00                  CP 0             ;test de
43106  C8                     RET Z            ;
43107  FE,19                  CP 25d           ;  validite
43109  D0                     RET NC           ;
43110  CD,1A,BC               CALL BC1A        ;position ecran
43113  7C                     LD A,H           ;point
43114  C6,38                  ADD A,38h        ;  sur derniere
43116  67                     LD H,A           ;  ligne
43117  E5                     PUSH HL
43118  EB                     EX DE,HL         ;dans DE
43119  E1                     POP HL
43120 01,50,00                LD BC,50h
43123  A7                     AND A            ;Ci=0
43124  ED,42                  SBC HL,BC        ;rangee precedente
43126  C3,08,A8               JP 43016         ;saut a ROU
```

Nous employons une routine du système Amstrad, située à BC1A. A l'entrée, H doit contenir le numéro de colonne −1 et L le numéro de rangée −1. A la sortie HL contient l'adresse d'écran : adresse de l'octet supérieur gauche du caractère défini à l'entrée. Cette routine ne fait aucun test de validité.

Pour cette raison, nous testons si le numéro de rangée est compris entre 1 et 24. Toute autre valeur provoque un retour sans exécution.

Après avoir ajouté 3800h à HL pour pointer sur la dernière ligne de la rangée sélectionnée, on transfère HL dans DE ; puis on fait pointer HL sur la dernière ligne de la rangée précédente en retranchant 50h.

HL et DE étant définis en tenant compte de l'offset d'écran éventuel (ce qui fait la routine du système), on saute dans notre programme précédent à 43016.
Entrez le BASIC :

```
10 INPUT "Quelle rangee voulez-vous agra
ndir";a
20 POKE 43000,a:POKE 43001,0
30 CALL 43100
40 END
```

RUN puis entrez 1 par exemple.

Si vous voulez agrandir la partie inférieure de l'agrandissement, faites RUN puis entrez 2.

LA SCRUTATION DU CLAVIER ET LES PAUSES

Les pauses illimitées, qui attendent l'enfoncement d'une touche pour orienter la suite du programme, et les pauses à durée limitée, pour ralentir ou faire varier le déroulement d'un programme, sont essentielles dans la construction d'un jeu interactif, ainsi que dans de nombreux autres programmes.

□ *Pause illimitée*

La pause qui attend l'enfoncement d'une touche se réalise à l'aide d'une routine système, située à BB06. A la sortie, A contient le caractère de la touche enfoncée.
Entrez le programme suivant :

```
43010  CD.06.BB          CALL BB06
43013  32.F8.A7          LD (43000).A
43016  C9                RET
```

puis le BASIC :

```
5 CLS
10 CALL 43010:a=PEEK(43000)
20 PRINT "touche enfoncee ":a:"code de "
:CHR$(a)
30 GOTO 10
90 END
```

Faites RUN et l'écran reste vierge tant que vous n'appuyez pas sur une touche.
Appuyez sur les touches, avec et sans SHIFT, pour voir leurs codes.
Si vous appuyez sur CTRL et la petite touche ENTER, vous avez la petite chaîne bien connue.

□ *Fonction INKEY$*

Le clavier est scruté 50 fois par seconde par le système. La fonction INKEY$ n'attend pas ; elle saisit une touche au passage. La routine système à employer, pour réaliser ceci, est située à BB09.

Transformez la ligne 43010 comme suit :

```
43010  CD,09,BB                    CALL BB09
```

et voyez la différence.
Ceci permet d'orienter la suite du programme suivant la touche enfoncée. Nous en montrons un exemple dans le jeu qui termine cette partie.

□ *Les pauses limitées*

Pour programmer des pauses à temps limité, nous employons une ou plusieurs boucles qui contiennent des instructions, au moins deux. Chaque instruction dure un certain temps, un certain nombre de cycle d'horloge T. Avec une horloge de 4 MHz, sa période T dure 0,25 microseconde.

Le délai programmé peut être très précis. Nous donnons dans les exemples suivants les nombre de périodes T que durent les instructions les plus couramment employées pour programmer un retard.

```
06,20          LD B,32        ;7T
10,FE         ↳DJNZ -2+2      ;13T quand B≠0
                                8T quand b=0
```

Ce programme boucle 32 fois sur lui-même. Sa durée est :

$7 + 31 \times 13 + 8 = 418T \times 0{,}25 = 104{,}5$ microsecondes.

On peut intercaler une instruction dans la boucle pour augmenter le temps :

```
06,20          LD B,32        ;7T
DD,BE,00     ┌→CP (IX+0)      ;19T
10,FB        └─DJNZ -5+2      ;13T pour B≠0
                                8T pour B=0
```

ce qui ajoute 32 × 19T soit 608T ou 152 microsecondes.
On peut employer deux boucles imbriquées :

```
06,20            LD B,32        ;7T
0E,FF        ┌──→LD C,255       ;7T
0D           │ ┌→DEC C          ;4T
20,FD        │ └─JR NZ,-3+2     ;12T si C≠0
             │                    7T SI C=0
10,F9        └───DJNZ -7+2      ;13T si B≠0
                                  8T si B=0
```

La boucle C dure $254 \times (4 + 12) + 4 + 7 = 4075T$.

La boucle totale dure 7 + 31 × (7 + 4075 + 13) + (7 + 4075 + 8) = 131042T ou 32760,5 microsecondes, soit 33 millisecondes.

L'emploi d'un registre double amène

```
21,00,50        LD HL,20480      ;10T
2B           ┌→DEC HL           ;6T
7C           │  LD A,H          ;4T
B5           │  OR L            ;4T
20,FB        └─JR NZ,-5+2       ;12T si Z=0
                                  7T si Z=1
```

Cette boucle dure 10 + 20479 × 26 + 21 = 532485T ou 133 millisecondes.
En faisant varier les chargements de B, C et HL dans les exemples précédents, vous obtenez les retards que vous désirez.
Les délais programmés nous serviront pour ralentir un jeu.
La temporisation est en effet nécessaire, car l'assembleur est trop rapide, parfois, pour qu'on ait le temps de voir ce qui se passe à l'écran.

LES NOMBRES ALÉATOIRES

Il existe plusieurs procédés pour générer des nombres pseudo-aléatoires. Nous disons pseudo car cette génération résulte d'une formule mathématique. Les tests faits sur les suites de nombres pseudo-aléatoires montrent des caractéristiques communes avec des nombres entièrement dûs au hasard.

La simulation de la vie, la création d'objet sur l'écran, etc., font appel à ces nombres. Leurs créations par programmation en C.M. nous évitera de devoir revenir au BASIC pour les créer et les saisir avec la fonction RND.

□ *Nombre aléatoire de 0 à 255*

Appliquons la formule :

$$A_{x+1} = 9.A_x + 1$$

Le nombre source A_x est stocké, par exemple, dans l'octet d'adresse 43000. On lui fait subir la transformation qui donne naissance au nombre suivant A_{x+1} que l'on stocke.
Voici le programme :

```
43010  3A,F8,A7     ALEA1   LD A,(43000)
43013  4F                   LD C,A
43014  87                   ADD A,A          ;2 fois A
```

```
43015   87                          ADD A,A           ;4 fois A
43016   87                          ADD A,A           ;8 fois A
43017   81                          AFF A,C           ;9 fois A
43018   3C                          INC A             ;plus 1
43019   32,F8,A7                    LD (43000),A
43022   C9                          RET
```

Comme nous aurons besoin de ce programme, sauvez-le sur cassette :

SAVE "JEU",b,43000,100

Pour voir son action, entrez le BASIC :

```
10 MODE 1
20 FOR n=1 TO 100
30 CALL 43010
40 PRINT PEEK(43000);
50 NEXT
90 END
```

Le programme génère 256 nombres, tous différents, après quoi la séquence recommence. Les opérations se font modulo 256. Pour voir les 100 premiers nombres, faites RUN. Le nombre pseudo-aléatoire se trouve à l'adresse 43000. Pour entrer n'importe où dans cette séquence, vous chargez, au préalable, l'octet d'adresse 43000 avec un nombre quelconque, plus petit que 256 évidemment.

Notez que cette suite génère des nombres alternativement pair, impair, ... et que son emploi pour certains jeux, comme la roulette par exemple, nécessite un aménagement. Pour notre jeu, dans lequel nous voulons afficher des objets au hasard sur l'écran, nous avons besoin d'un nombre pour la colonne et d'un autre nombre pour la rangée. Ecrivons un deuxième programme, en employant la formule :

$A_{x+1} = 13.A_x + 1$

Entrez le programme :

```
43023   3A,F9,A7        ALEA2       LD A,(43001)
43026   4F                          LD C,A
43027   87                          ADD A,A           ;2 fois A
43028   81                          ADD A,C           ;3 fois A
43029   87                          ADD A,A           ;6 fois A
```

```
43030  87         ADD A,A         ;12 fois A
43031  81         ADD A,C         ;13 fois A
43032  3C         INC A           ;plus 1
43033  32,F9,A7   LD (43001),A    ;le stocke
43036  C9         RET
```

Sauvez les 2 programmes, ALEA1 et ALEA2, sur cassette :

SAVE "JEU",b,43000,100

En combinant les deux, vous générez des nombres entre 0 et 65535. Changez deux lignes BASIC :

```
30 CALL 43010:CALL 43023
40 PRINT PEEK(43000)+256*PEEK(43001);
```

□ *Nombre de 0 à 65535*

Cette façon de procéder ne permet pas d'avoir une suite de 65536 nombres différents. Si vous désirez une telle suite, voici un programme qui utilise HL et emploie la formule :

$A_{x+1} = 29.A_x + 1$

```
43037  2A,FA,A7   LD HL,(43002)
43040  E5         PUSH HL
43041  D1         POP DE          :DE=HL
43042  29         ADD HL,HL       :2 fois HL
43043  19         ADD HL,DE       :3 fois HL
43044  29         ADD HL,HL       :6 fois HL
43045  19         ADD HL,DE       :7 fois HL
43046  29         ADD HL,HL       :14 fois HL
43047  29         ADD HL,HL       :28 fois HL
43048  19         ADD HL,DE       :29 fois HL

43049  23         INC HL          :plus 1
43050  22,FA,A7   LD (43002),HL   :le stocke
43053  C9         RET
```

Ne sauvez pas ce programme sur cassette, du moins pour notre jeu.

Si vous désirez voir la suite de nombres, entrez :

```
10 CLS
20 FOR n=1 TO 100
30 CALL 43037
40 PRINT PEEK(43002)+256*PEEK(43003);
50 NEXT
60 GOTO 20
90 END
```

Quand vous voulez examiner l'écran, appuyez sur ESC une fois.

AFFICHAGE D'UNE BORDURE

Affichons une bordure sur les quatre côtés de l'écran, en employant de l'encre rouge et le caractère de code 127 (voyez son dessin en annexe A3 de votre guide). Ce sera l'occasion de vous montrer un programme qui s'appelle lui-même et qui se modifie lui-même.

Voici le programme :

```
43100  3E,03      COTES   LD A,3          ;rouge
43102  CD,90,BB           CALL BB90       ;met la plume rouge
43105  3E,24              LD A,24h        ;C.M. de INC H
43107  32,8D,A8           LD (43149),A    ;modifie 43149
43110  21,01,01           LD HL,0101      ;col. 1, rangee 1
43113  06,28              LD B,40d        ;40 affichages
43115  CD,88,A8           CALL 43144      ;routine AFF
43118  21,19,01           LD HL,0119      ;col. 1, rangee 25
43121  06,28              LD B,40d        ;40 affichages
43123  CD,88,A8           CALL 43144      ;routine AFF
43126  3E,2C              LD A,2C         ;C.M. de INC L
43128  32,8D,A8           LD (43149),A    ;modifie 43149

43131  21,01,01           LD HL,0101      ;col. 1, rangee 1
43134  06,19              LD B,25d        ;25 affichages
43136  CD,88,A8           CALL 43144      ;pour afficher
43139  21,01,28           LD HL,2801      ;col. 40, rangee 1
43142  06,19              LD B,25d        ;25 affichages
43144  E5         AFF   ┌►PUSH HL
43145  CD,75,BB         │ CALL BB75       ;position curseur
43148  E1               │ POP HL
43149  24               │ INC H           ;ou  INC L
43150  3E,7F            │ LD A,127d       ;carac. graphique
43152  CD,5A,BB         │ CALL BB5A       ;vers l'ecran
43155  10,F3            └─DJNZ -13+2
43157  C9                 RET
```

Pour voir la bordure, entrez le BASIC :

```
10 MODE 1
20 CALL 43100
30 IF INKEY$="" THEN 30
40 PEN 1
90 END
```

RUN et voyez la bordure.

Sauvez ce programme en C.M., avec les 2 programmes ALEA,

SAVE "JEU",b,43000,200

La routine système BB90 met la couleur de plume. A l'entrée, A doit contenir la couleur. Les lignes 43100 et 43102 correspondent à PEN 3. Nous devons rester en mode 1.

Pour afficher les deux côtés horizontaux, nous devons incrémenter le numéro de colonne, d'où le placement du C.M. 24 dans la ligne 43149 ; et pour afficher les deux côtés verticaux, nous entrons le C.M. 2C dans la ligne 43149 pour incrémenter le numéro de rangée. Les trois premiers côtés sont affichés en appelant la sous-routine AFF et le quatrième côté poursuit dans cette routine.

La routine système BB75, pour positionner le curseur, et la routine BB5A, pour afficher un caractère, ont déjà été employées dans les parties précédentes.

Pour cette routine, comme pour les suivantes, et les deux précédentes (ALEA), vous devez évidemment les recharger si vous avez débranché votre ordinateur entre-temps.

AFFICHAGE D'OBJETS AU HASARD

Nous affichons 80 objets, au hasard sur l'écran ; les objets sont des caractères de code 238 (voyez l'annexe A3 de votre guide) sur fond turquoise vif. Avant l'affichage de chaque objet, nous générons un couple, numéro de colonne et numéro de rangée, de manière que l'objet soit à l'intérieur des bordures dessinées dans le programme précédent.

```
43158  3E,02        OBST        LD A,2          ;turquoise vif
43160  CD,96,BB            ┌───→CALL BB96       ;met le papier
43163  06,50               │    LD B,80d        ;80 affichages
43165  CD,02,A8            │    CALL 43010      ;ALEA1
43168  FE,26               ↑  ┌─CP 38d
```

```
43170  38,04      JR C,4+2
43172  D6,26      SUB A,38d      ;A= de 0 a 37
43174  18,F8      JR -8+2
43176  3C         INC A          ;A= de 1 a 38
43177  3C         INC A          ;A= de 2 a 39
43178  67         LD H,A         ;numero de colonne
43179  CD,0F,A8   CALL 43023     ;ALEA2
43182  FE,17      CP 23d
43184  38,04      JR C,4+2
43186  D6,17      SUB A,23d      ;A= de 0 a 22
43188  18,F8      JR -8+2
43190  3C         INC A          ;A= de 1 a 23
43191  3C         INC A          ;A= de 2 a 24
43192  6F         LD L,A         ;numero de rangee
43193  CD,75,BB   CALL BB75      ;pos. affichage
43196  3E,EE      LD A,238d      ;objet
43198  CD,5A,BB   CALL BB5A      ;l'affiche
43201  10,DA      DJNZ -38+2
43203  C9         RET
```

Sauvez ce programme avec les précédents :

SAVE "JEU",b,43000,300

Pour visualiser, entrez le BASIC :

```
10 MODE 1
20 CALL 43100:CALL 43158
30 IF INKEY$="" THEN 30
40 PEN 1:PAPER 0
50 END
```

Faites plusieurs fois RUN.

Les lignes 43158 et 43160 correspondent à PAPER 2.
Les deux routines ALEA, vues précédemment, retournent un nombre aléatoire dans A ; les registres A et C sont corrompus. Notez que C contient également le nombre aléatoire précédent. Après l'appel de ALEA1, nous devons réduire le nombre aléatoire en dessous de 38, d'où les soustractions successives éventuelles pour amener A à une valeur de 0 à 37. Après deux incrémentations, A contient un nombre de 2 à 39 que l'on charge dans H comme numéro de colonne.
Nous faisons de même, en appelant ALEA2, pour fixer le numéro de rangée entre 2 et 24.

Le reste ne présente pas de difficultés.

4

Jeu d'évitement

Le jeu consiste à déplacer un bonhomme, à l'aide des quatre touches fléchées, de façon à éviter les obstacles. En plus des routines précédentes, nous avons besoin d'une routine d'affichage pour afficher plusieurs messages. Ecrivons-la :

```
43037  7E         AFFI  ┌→LD A,(HL)
43038  FE,FF            │ CP FF
43040  C8               │ RET Z
43041  CD,5A,BB         │ CALL BB5A       ;affiche l'octet A
43044  23               │ INC HL
43045  18,F6            └─JR -10+2
```

Cette petite routine n'a rien de spécial. Avant l'appel, HL doit contenir l'adresse du message à afficher. De plus ce message doit inclure la position d'affichage, à l'aide des codes de contrôle que nous avons déjà vus.

Nous avons besoin de deux messages : le premier au début de la partie pour démarrer le score et le second en fin de partie.

Entrez les C.M. suivants :

```
43050  1F.0F.19               DEF colonne 15, rangee 25
43053  50,4F.49.4E            DEF POIN
43057  54.53,3D               DEF TS=
43060  30.30.30.FF            DEF 000.fin de message

43064  1F.0B.01               DEF colonne 11, rangee 1
43067  56,6F,75,73,20         DEF Vous_
43072  61,76,65.7A,20         DEF avez_
43077  74.72,65.62,75         DEF trebu
43082  63.68,65.21            DEF che!
43086  FF                     DEF fin de message
```

Ecrivons un programme pour afficher le score initial :

```
43210  3E.00      SCORE   LD A.0
43212  CD.90.BB           CALL BB90        :plume bleue
43215  3E.01              LD A.1
43217  CD.96,BB           CALL BB96        ;papier jaune
43220  21.2A.A8           LD HL.43050      :adresse message
43223  CD.1D.A8           CALL 43037       :AFFI
43226  C9                 RET
```

Sauvez tous ces programmes sur cassette :

SAVE "JEU",b,43000,400

Dès à présent, mettez 400 comme nombre d'octets.
Pour visualiser, faites CALL 43210.

Avant d'en venir au jeu proprement dit, écrivons une routine pour modifier le score à chaque pas réussi par le bonhomme :

```
43227  CD.9C.BB   MANIP   CALL BB9C        :inv. plume papier
43230  21.3C.A8           LD HL.43062      :adresse unites
43233  7E                 LD A,(HL)        ;saisie
43234  FE.39              CP 39h           ;est-ce chiffre 9?
43236  30.03             ┌JR NC.3+2        :si oui, saut
43238  34                ↓INC (HL)         :incre. les unites
```

```
43239  18.0F          ┌┬JR 15+2
43241  36.30          │└►LD (HL).30          :unites=0
43243  2B             │  DEC HL              :adresse dizaines
43244  7E             │  LD A,(HL)
43245  FE.39          │  CP 39               :est-ce 9?
43247  30.03          │┌─JR NC.3+2           :si oui,saut
43249  34             ││ INC (HL)
43250  18.04          ●┼─JR 4+2
43252  36.30          │└►LD (HL).30          :dizaines =0
43254  2B             │  DEC HL              :adresse centaines
43255  34             │  INC (HL)
43256  21.2A,A8       └─►LD HL.43050         :adresse message
43259  CD.1D.A8          CALL 43037          :AFFI
43262  CD.9C.BB          CALL BB9C           :inverse
43265  C9                RET
```

Nous appelons deux fois la routine système BB9C pour inverser les couleurs plume et papier.

Le reste du programme manipule le score. On extrait le chiffre des unités ; si ce chiffre n'est pas 9, on l'incrémente, puis on saute à l'affichage ; si ce chiffre est 9, on met 0 dans les unités, puis on passe au chiffre des dizaines. Ce chiffre subit le même traitement que celui des unités. Nous n'avons rien prévu quand le chiffre des centaines atteint 9, car cela n'arrive jamais.

N'oubliez pas :

SAVE "JEU",b,43000,400

ROUTINE JEU

Voici le plan de cette routine :

- Affichage au centre de l'écran d'un bonhomme de code 248.
- Boucle d'attente, pour permettre au joueur d'examiner l'écran afin de décider le chemin à parcourir.
- Délai programmé pour ralentir le jeu.
- Saisie d'une des quatre touches fléchées et changement de la position du bonhomme.
- Si cette position est un obstacle, le jeu se termine.
- Si non, affichage du bonhomme dans sa nouvelle position, puis augmentation du score.
- Saut à Délai.

Voici le programme :

```
43300  CD,9C,BB   JEU     CALL BB9C      ;inverse
43303  21,0C,14           LD HL,140C     ;col. 20,rangee 12
43306  CD,75,BB           CALL BB75      ;pos. affichage
43309  3E,F8              LD A,248d      ;bonhomme
43311  CD,5A,BB           CALL BB5A      ;l'affiche
43314  CD,06,BB   ATT.    CALL BB06      ;attend un carac.
43317  32,FA,A7           LD (43002),A   ;le stocke
43320  01,00,80   DELAI   LD BC,8000h
43323  0B                 DEC BC
43324  78                 LD A,B
43325  B1                 OR C
43326  20,FB              JR NZ,-5+2
43328  CD,09,BB   SAIS.   CALL BB09      ;lit carac. clavier
43331  30,03              JR NC,-3+2
43333  32,FA,A7           LD (43002),A   ;le stocke
43336  CD,78,BB   CUR.    CALL BB78      ;lit pos. curseur
43339  3A,FA,A7           LD A,(43002)   ;touche dans A
43342  25                 DEC H

43343  FE,F0              CP F0          ;↑
43345  20,01              JR NZ,1+2
43347  2D                 DEC L
43348  FE,F1              CP F1          ;↓
43350  20,01              JR NZ,1+2
43352  2C                 INC L
43353  FE,F2              CP F2          ; ←
43355  20,01              JR NZ,1+2
43357  25                 DEC H
43358  FE,F3              CP F3          ; →
43360  20,01              JR NZ,1+2
43362  24                 INC H
43363  CD,75,BB   POSCU   CALL BB75      ;pos. affichage
43366  CD,60,BB   CARAC   CALL BB60      ;lit carac. d'ecran
43369  FE,20              CP 32d         ;code espace
43371  C0                 RET NZ         ;retour si obstacle

43372  3E,F8              LD A,248d      ;bonhomme
43374  CD,5A,BB           CALL BB5A      ;l'affiche
43377  CD,78,BB   MANI    CALL BB78      ;lit pos. affichage
43380  00                 NOP
43381  E5                 PUSH HL        ;la sauve
43382  CD,DB,A8           CALL 43227     ;MANIP
43385  E1                 POP HL         ;la recupere
43386  CD,75,BB           CALL BB75      ;positionne aff.
43389  18,B9              JR -71+2
```

Sauvez sur cassette :

SAVE "JEU",b,43000,400

Nous commençons par inverser les couleurs plume et papier avant d'afficher le bonhomme au centre de l'écran. Le bonhomme doit s'afficher sur papier bleu. En effet, quand on fait la comparaison en 43369, tout ce qui est bleu est considéré comme des espaces.

La boucle ATT. attend l'enfoncement d'une touche pour vous permettre d'examiner l'écran. Dès que vous avez appuyé sur une touche, on stocke son code dans 43002.
On entre ensuite dans un retard programmé pour ralentir le jeu.
A l'étiquette SAIS. on lit le caractère du clavier comme le fait INKEY$. A la sortie de la routine système BB09, l'indicateur C égale 0 si aucune touche n'a été pressée. Le saut en 43331 évite qu'une valeur erronée de A soit mise dans 43002.

A l'étiquette CUR. on lit la position du bonhomme, puis on charge A avec la valeur de la touche pressée, ou avec la valeur précédente, si on n'a enfoncé aucune touche. Ensuite on fait DEC H pour diminuer le numéro de colonne, car la ligne 43374 qui affiche le bonhomme, incrémente aussi le numéro de colonne (vous l'avez remarqué dans nos programmes d'affichage). On compare ensuite A aux quatre codes des touches fléchées et, suivant le cas, on incrémente ou décrémente le numéro de colonne ou de rangée.

A l'étiquette POSCU, on donne la nouvelle position du bonhomme. Puis, en CARAC, on lit le caractère d'écran qui se trouve à cette nouvelle position. Si ce n'est pas un espace, il y a retour, sinon on affiche le bonhomme.

Pour finir, on incrémente le score en appelant MANIP. Avant cet appel, on sauve la position d'affichage du bonhomme dans la pile, puis on la restaure après l'appel. Finalement, on saute à DELAI pour le tour suivant.

ROUTINE D'INITIALISATION

Pour exploiter toutes ces routines, écrivons une routine d'initialisation :

```
43267  CD,14,BC     INIT   CALL BC14       :CLS
43270  21,34,A8            LD HL,43060     :pour remettre
43273  36,30               LD (HL),30      :  les trois
43275  23                  INC HL          :  chiffres du
43276  36,30               LD (HL),30      :  compteur
43278  23                  INC HL          :  a
43279  36,30               LD (HL),30      :  zero

43281  CD,5C,A8            CALL 43100      :bordure
43284  CD,96,A8            CALL 43158      :obstacles
43287  CD,CA,A8            CALL 43210      :affiche message
43290  CD,24,A9            CALL 43300      :jeu
43293  21,38,A8            LD HL,43064     :adresse message
43296  CD,1D,A8            CALL 43037      :l'affiche
43299  C9                  RET
```

N'oubliez pas : SAVE "JEU",b,43000,400
Comme tout est en C.M., faites la commande BASIC :

CALL 43267

Vous avez le bonhomme au centre de l'écran. Observez bien cet écran pour déterminer le chemin que va suivre le bonhomme pour faire le plus grand nombre de pas possibles en évitant les obstacles. Vous faites progresser le bonhomme à l'aide des quatre touches fléchées. Si vous enfoncez une autre touche, le jeu s'arrête. Dès que le jeu est commencé, vous ne pouvez plus l'arrêter, sauf si le bonhomme bute sur un obstacle : un bord de l'écran, un objet ou une trace de son passage qu'il ne peut franchir.
Essayez d'avoir un score élevé.

Si vous désirez moins d'obstacles, ou plus, sur l'écran, modifiez la ligne 43163 en faisant, par exemple :

POKE 43164,40	pour la facilité, ou
POKE 43164,128	pour la difficulté.

Si vous désirez que le bonhomme se déplace plus vite, diminuez le délai de la ligne 43320, en faisant, par exemple :

POKE 43322,32

Le déplacement le plus lent s'obtient en faisant :

POKE 43322,0

En agissant sur ces deux paramètres : vitesse et nombre d'obstacles, vous augmentez votre plaisir de jouer.
Bon amusement.

LE SYSTÈME AMSTRAD | 6

Nous donnons les adresses d'entrée des routines du système ainsi que leurs exigences. Certaines de ces routines ont déjà été exploitées au cours des parties précédentes. Nous illustrons, ci-après, certaines de ces routines par des exemples. Nous disons certaines, car il y en a des centaines.

Le système Amstrad, qui est très élaboré, permet une multitude d'applications possibles. Nous examinons succinctement ce système. Les lecteurs intéressés peuvent se procurer le livre *The complete firmware specification – CPC 464* (soft 158) chez Amstrad.

1

Le plan mémoire

Le plan de la mémoire est représenté figure 1.

Il y a 64 K. d'octets de MEV (RAM), d'adresses 0 à FFFF, et 32 K. d'octets de MEM (ROM). Comme le microprocesseur ne peut adresser que 64 K. avec ses 16 lignes d'adresses, les ROM ont des adresses communes avec celles de la RAM.

On distingue deux ROM de 16 K. chacune. La ROM inférieure, d'adresses 0 à 3FFF, contient le système Amstrad et son système opératoire. La ROM supérieure, d'adresses C000 à FFFF, contient le système BASIC Amstrad.

Quand on écrit dans la mémoire, c'est toujours dans la RAM puisque les ROM ne peuvent s'altérer. En lecture, vous lisez la RAM avec les instructions BASIC. Vous pouvez lire la ROM, en construisant un programme en C.M. avec emploi du système opératoire.

Le système opératoire peut commuter jusqu'à 251 autres ROM supérieures, en plus de la ROM BASIC incorporée, et permet l'emploi de quatre ROM supérieures simultanément.

Les ROM ont besoin de mémoire vive pour leur fonctionnement ; elles y stockent des données variables, les manipulent pour les mettre à jour, etc. La ROM inférieure dispose, en haut de mémoire RAM, des octets d'adresses B100 à BFFF.

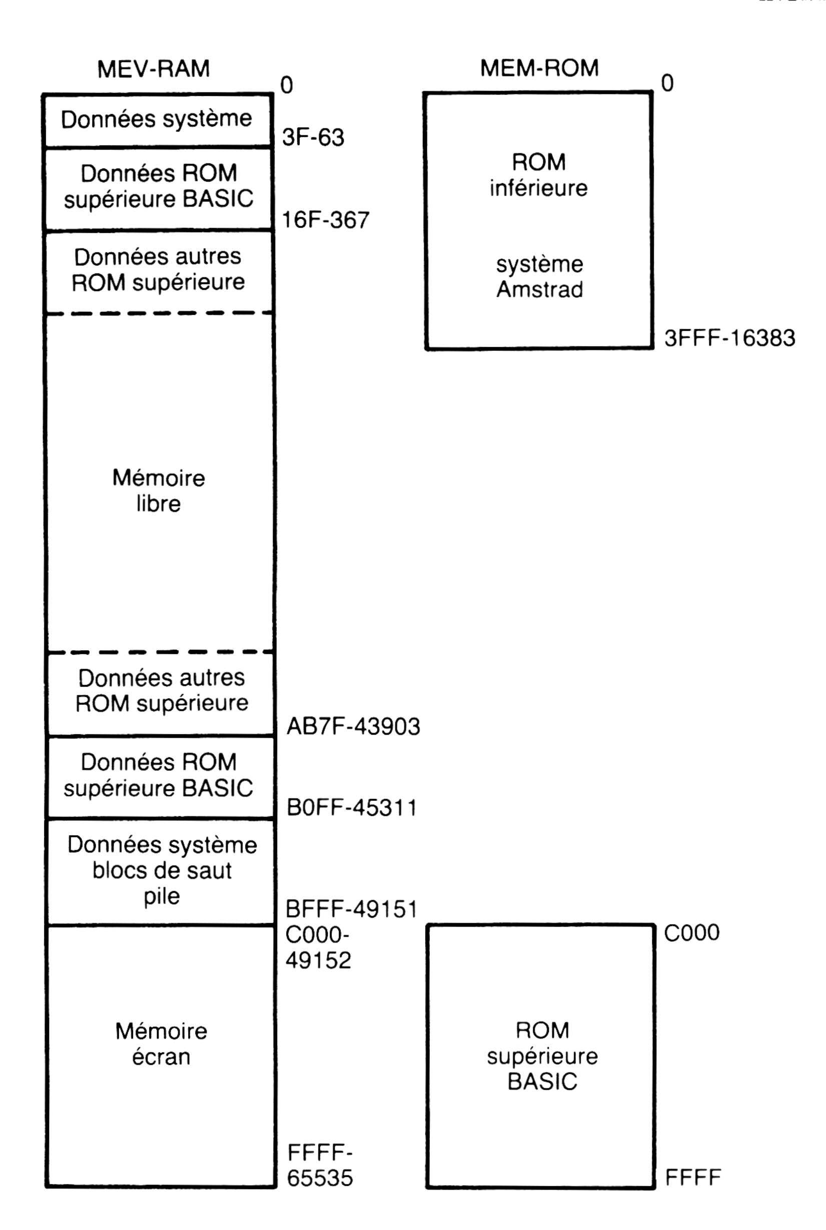


Figure 1. — Plan de la mémoire

Elle y loge des variables système, des blocs de sauts, des routines et la pile du système qui est contiguë à la zone d'écran. La pile commence à BFFF et s'étend vers le bas d'au moins 256 octets. D'autre part, la ROM inférieure dispose, en bas de mémoire RAM, des octets d'adresse 0 à 3F, pour l'usage du système opératoire.

La ROM supérieure dispose, en haut de mémoire, d'une zone allant de AB80 à B0FF, pour ses variables système et ses tampons, et, en bas de mémoire, d'une zone allant de 40 à 16F. Si on ajoute des ROM supérieures, appelées également ROM latérales, sélectionnées par commutation, elles peuvent également s'adjuger des zones de RAM.

La zone centrale constitue la mémoire libre pour l'utilisateur. En fait, cette zone est utilisée par le BASIC pour y loger des programmes BASIC et les variables associées. Pour notre propre usage (logement de C.M. par exemple), on réserve notre zone de travail avec la commande MEMORY, et une zone d'adresses AB7F - 43903, et en dessous, est à notre disposition.

Quand l'ordinateur vient d'être mis sous tension, faites : PRINT HIMEM qui donne 43903, qui est l'adresse supérieure à ne pas dépasser. Faites ensuite : PRINT FRE(0) qui donne 43533 et 43903 - 43533 soit 370 qui est l'adresse inférieure à ne pas surpasser. En fait, un programme BASIC commence à 170h - 368d.

2

Le système opératoire et ses blocs de sauts

LE SYSTÈME OPÉRATOIRE

Le système Amstrad, situé dans la ROM inférieure, contient plusieurs sections de traitement. Ces sections s'occupent du clavier, de l'écran, de l'enregistreur, des sons et du système opératoire. Celui-ci s'appelle KERNEL, Nous conservons ce mot anglais qui pourrait se traduire par graine ou semence du système.

Le KERNEL s'occupe des interruptions, des événements, de la sélection des ROM externes et internes et du déroulement des programmes. Il comporte plusieurs zones d'accès possibles.

Examinons, tout d'abord, la zone qui emploie les instructions RST vues au début de la partie précédente et qui accèdent au début de la mémoire. Comme nous l'écrivions, les instructions RST n'ont qu'un octet de C.M. Cependant, le système les emploie autrement, à l'exception de la première RST 0, de C.M. C7. Pour les différencier, le système appelle les RST suivants : RST 1, RST 2, ..., RST7 au lieu de RST 8, RST 10, ..., RST 38. Nous signalons ceci car les Assembleurs du commerce ne tiennent pas tous compte de cette particularité ; tous les désassembleurs non plus d'ailleurs.

Comme les différentes entrées de cette zone doivent être accessibles, peu importe l'état de la ROM inférieure : disponible ou indisponible, cette zone est recopiée en RAM au cours de l'initialisation aux adresses 0-3F, 0-63d. L'usager ne doit pas modifier ces octets, sinon des comportements imprévisibles peuvent survenir, jusqu'au plantage de l'ordinateur.

RST 0 : provoque le redémarrage de l'ordinateur, comme lorsque celui-ci a été mis sous tension.

```
CD,00,00        CALL 0
C3,00,00        JP 0
C7              RST 0
```

Ces trois instructions provoquent l'initialisation. Elles ont le même effet que CTRL-SHIFT-ESC poussées ensemble.

RST 1 : ce C.M., CF, doit être suivi de deux octets qui contiennent l'adresse à laquelle un saut est effectué. Cette adresse a une particularité. Comme la routine vers laquelle on saute se trouve dans la ROM inférieure, qui s'étend jusqu'à 3FFF - 16383, on n'a besoin que de 14 bits pour définir cette adresse ; de ce fait les bits 15 et 14 ont une signification spéciale.

15	14	13 ... 2 1 0
		adresse

Le bit 15 à 1 rend la ROM supérieure indisponible.
Le bit 14 à 1 rend la ROM inférieure indisponible.

Par exemple, la routine BB5A, que nous avons employée à plusieurs reprises pour afficher un caractère sur l'écran :

```
CD,5A,BB        CALL BB5A
```

Cette routine contient ceci :

```
BB5A CF,00,94   JP 1400h
```

Cela correspond à un saut à 1400h, soit 9400h-8000h (bit 15). Le bit 15 à 1 met la ROM supérieure hors service, le bit 14 à 0 met la ROM inférieure en service, l'adresse réelle est 1400h. Tous les registres passent tels qu'ils sont dans la routine. Celle-ci n'emploie que le ou les registres dont elle a besoin. Au retour, la routine met certains registres pour transmettre des données.

Quand le bit 14 est à 1, la ROM inférieure est hors service et le saut se fait en RAM.
Cette instruction "Amstrad" ne s'emploie que pour des adresses inférieures à 3FFFh.
B : fonctionne comme RST 1, mais l'adresse de la routine se trouve dans HL. Les bits 15 et 14 ont les mêmes significations qu'avec RST 1. L'adressage possible ne comporte que 14 bits.

```
21,00,94        LD HL,9400
CD,0B,00        CALL Bh
```

appellent la routine 1400h comme ci-dessus.

RST 2 et RST 3 : ces instructions provoquent un saut à une routine située dans une ROM supérieure, latérale. Elles sont de très peu d'intérêt pour votre usage.

RST 4 : correspond à LD A,(HL). HL contient l'adresse mémoire à lire. Il s'agit toujours d'une adresse en RAM, les ROM sont inactives.

RST 5 : pour sauter dans une routine de la ROM inférieure. Le C.M. EF remplace le premier C.M. d'une instruction de saut C3,qq,pp.

RST 6 : est une instruction de redémarrage, disponible pour l'utilisateur. Les octets 30h à 37h doivent contenir la routine de l'utilisateur.

RST 7 : Le microprocesseur Z 80 supporte trois modes d'interruption. Au cours de l'initialisation, il est mis dans le mode 1 d'interruption. Dans ce mode, chaque interruption provoque un RST 7. Il y a 50 interruptions par seconde, à chaque synchro trame du signal vidéo, qui provoquent un RST 7, donc un saut à une routine d'adresse 38h. Cette routine effectue, entre autres, la scrutation du clavier.

BLOC DE SAUTS RÉSERVÉ AU KERNEL

Ce bloc de sauts accède à des routines KERNEL qui traitent de la sélection et de l'état des ROM. Toutes ces routines sont en mémoire RAM ; elles y ont été copiées au cours de l'initialisation. Ceci permet leur utilisation, tandis que la sélection et l'état d'une ROM sont changés. Les points d'entrée vont de B900 à B921. Cette dernière entrée se situe directement dans la routine en C.M. Les autres entrées accèdent aux routines à l'aide d'une instruction de saut C3,qq,pp. Ces différentes routines se terminent à BAE8. L'usager ne doit changer aucun octet de cette zone. L'appel se fait avec CALL. Par exemple pour la première routine, CALL,00,B9.

- *B900 :* Rend disponible la ROM supérieure. A la suite de ceci, les lectures faites aux adresses C000 à FFFF, renvoient le contenu de la ROM supérieure.
  Entrée : Pas de condition.
  Sortie : A contient l'état précédent de la ROM ; F est indéfini.

- *B903 :* Rend indisponible la ROM supérieure. Les lectures faites par la suite, aux adresses C000 à FFFF, renvoient le contenu de la RAM, soit la mémoire d'écran.
  Entrée : Pas de condition.
  Sortie : A contient l'état précédent de la ROM ; F est indéfini.

- *B906 :* Rend disponible la ROM inférieure. Les lectures aux adresses 0 à 3FFF, renvoient le contenu de la ROM.
  Entrée : Pas de condition.
  Sortie : A contient l'état précédent de la ROM ; F est indéfini.

- *B909 :* Rend la ROM inférieure indisponible. Les lectures aux adresses 0 à 3FFF se font en RAM.
  Entrée : Pas de condition.
  Sortie : A contient l'état précédent de la ROM ; F est indéfini.

- *B90C :* Restaure l'état précédent de la ROM. A employer après l'appel d'une des quatre routines précédentes, après chargement de A.
  Entrée : A contient l'état précédent de la ROM.
  Sortie : A et F indéfinis.

- *B91B :* Réalise une instruction LDIR et rend indisponibles les deux ROM. Ceci permet de transférer des zones de RAM, quel que soit l'état des ROM.
  Entrée : BC, DE et HL mis comme pour l'instruction LDIR.
  Sortie : BC, DE, HL et F mis par l'exécution de LDIR.

- *B91E :* Réalise une instruction LDDR et rend indisponibles les deux ROM.
  Les conditions sont identiques à celles de la routine précédente.

3

Le bloc de sauts principal

Les autres sections traitées par le système sont accessibles en appelant l'une des entrées du bloc de sauts principal.

Ce bloc de sauts commence à l'adresse BB00 et se termine à BD39. Il occupe une zone de mémoire RAM, qui est recopiée de la ROM inférieure lors de l'initialisation. Chaque entrée est un groupe de trois octets, le premier étant CF (ce sont toutes des RST 1). Le deuxième et le troisième octets donnent une adresse de la ROM, inférieure, avec le bit 15 à 1 et le bit 14 à 0 pour rendre l'écran accessible. Etant situés en RAM, ces sauts peuvent être modifiés par l'utilisateur.

Le BASIC fait un usage fréquent de ce bloc de sauts. Par conséquent, si vous modifiez une adresse de saut, le BASIC se comportera différemment. De plus, chaque routine du système demande des conditions d'entrée et fournit des paramètres de sortie à l'aide des registres du microprocesseur : il faut tenir compte de ces éléments pour modifier une adresse de saut.

Après ce bloc de sauts principal, il y a un bloc de routines d'indirection, qui sont utilisées par les routines du système, définies plus haut. Tout changement d'indirection modifie le comportement des routines système. La description des indirections sort du cadre de cet ouvrage.

Le bloc de sauts principal est divisé en sections, que nous examinons maintenant. A l'exception des indicateurs d'état mentionnés, le reste du registre F est indéfini.

LE CLAVIER

Cette section s'occupe du clavier et des joystiks, ainsi que de leurs caractéristiques telles que : touches pressées, touches avec CRTL et SHIFT, touches de fonction, répétition des touches, autres valeurs attribuées aux touches, etc.

Il y a trois niveaux d'opération :
- la scrutation du clavier,
- l'attribution d'un numéro à la touche pressée,
- la conversion de ce numéro en un caractère suivant table de corrélation.

Caractères

□ *BB06 :* Attend le caractère suivant du clavier.
Entrée : Pas de condition.
Sortie : C_i=1 et A contient le caractère.

□ *BB09 :* Prend le caractère du clavier s'il y en a un. Cette routine n'attend pas.
Entrée : Pas de condition.
Sortie : S'il y a un caractère : C_i=1 et A contient le caractère.
s'il n'y en a pas : C_i=0 et A est indéfini.

□ *BB0C :* Retourne un caractère du clavier pour la fois suivante. Cette routine sauve un caractère pour le prochain appel de BB06 ou de BB09.
Entrée : A contient le caractère à sauver.
Sortie : Registres préservés.

□ *BB0F :* Met une chaîne d'expansion associée à un signe d'expansion.
Entrée : B contient le signe d'expansion.
C contient la longueur de la chaîne.
HL contient l'adresse de la chaîne.
Sortie : Si l'expansion est correcte C_i=1.
Si la chaîne est trop longue ou le signe d'expansion incorrect C_i=0.

Le signe d'expansion que doit contenir B est 0 à 31 ou 128 à 159 : c'est la même chose car la routine effectue un masquage avec 7F pour éliminer le bit 7. Il y a 32 signes d'expansion possibles. Les chaînes par défaut pour les 13 premiers signes, se trouvent dans votre guide, page A3.15.

□ *BB12 :* Obtient un caractère d'une chaîne d'expansion. Les caractères dans la chaîne sont numérotés à partir de 0.
Entrée : A contient le signe d'expansion.
L contient le n° du caractère de la chaîne.
Sortie : Si le caractère est trouvé, C_i=1 et A contient le caractère.

Si le signe est incorrect ou si la chaîne n'est pas assez longue C_i=0 et A est indéfini.
Les registres D et E sont indéfinis.

□ *BB15 :* Assigne un tampon pour des chaînes d'expansion. Initialise le tampon avec les chaînes d'expansion par défaut.
Entrée : DE contient l'adresse du tampon.
HL contient la longueur du tampon.
Sortie : Si le tampon est bon, C_i=1.
Si le tampon est trop court, C_i=0.
Tous les registres simples sont indéfinis.

Touches

□ *BB18 :* Attend la touche suivante du clavier.
Entrée : Aucune condition.
Sortie : C_i=1 et A contient le caractère ou le signe d'expansion.

□ *BB1B :* Prend une touche du clavier. La routine n'attend pas.
Entrée : Pas de condition.
Sortie : Si une touche est disponible C_i=1 et A contient le caractère ou le signe d'expansion.
S'il n'y a pas de touche, C_i=0 et A est indéfini.

□ *BB1E :* Teste si une touche est pressée ou un joystik actionné.
Entrée : A contient un numéro de touche (page A3.16 du guide).
Sortie : Si la touche est pressée, Z=0.
Si elle n'est pas pressée, Z=1, C_i=0 et C contient l'état courant de SHIFT et CTRL.
A, H et L sont indéfinis.

□ *BB21 :* Demande l'état de CAPS LOCK et SHIFT LOCK.
Entrée : Pas de condition.
Sortie : L contient l'état de SHIFT LOCK.
H contient l'état de CAPS LOCK.
L'état 0 signifie non enclenché.
L'état FF signifie enclenché.

□ *BB24 :* Demande l'état du ou des joysticks.
Entrée : Pas de condition.
Sortie : A contient l'état du joystick 0.
H contient l'état du joystick 0.
L contient l'état du joystick 1.
Les autres registres sont préservés.

L'état du joystick est signalé par les bits mis à 1.
bit 0 : dessus
bit 1 : dessous
bit 2 : gauche
bit 3 : droite
bit 4 : feu 2
bit 5 : feu 1
bit 6 : réserve
bit 7 : toujours 0

Table de corrélation

□ *BB27 :* Affecte un caractère ou un signe d'expansion à une touche simple dans la table de corrélation simple.
Entrée : A contient le numéro de touche.
B contient la nouvelle corrélation.
Sortie : A, H et L indéfinis.
Si le numéro de touche est incorrect, plus grand que 79, rien ne se passe. La nouvelle corrélation peut être un signe d'expansion 128 à 159.

□ *BB2A :* Demande la corrélation d'une touche simple.
Entrée : A contient le numéro de touche.
Sortie : A contient la corrélation de la touche simple.
H et L sont indéfinis.
Il n'y a pas de test de validité du numéro de touche ; la corrélation retournée est fausse si le numéro de touche est plus grand que 79.

□ *BB2D :* Affecte un caractère ...

□ *BB30 :* Demande la corrélation ... d'une touche shiftée (excusez l'anglicisme) dans la table de corrélation avec SHIFT.
Ces deux routines fonctionnent comme les deux précédentes mais une pression simultanée sur SHIFT et une autre touche.

□ *BB33 :* Affecte un caractère ...

□ *BB36 :* Demande la corrélation ... d'une touche avec CTRL dans la table de corrélation avec CTRL.
Fonctionnent comme les deux précédentes mais avec enfoncement simultané de CTRL et d'une autre touche.

Exemple 1 : Utilisons la routine BB12 pour afficher la première lettre de la chaîne associée au signe d'expansion 140. Nous savons que cette chaîne est RUN" mais ceci est un exemple.

```
43010  3E,8C          LD A,140       ;signe d'expansion
43012  2E,00          LD L,0         ;premier caractere
43014  CD,12,BB       CALL BB12      ;l'extrait
43017  CD,5A,BB       CALL BB5A      ;l'affiche
43020  C9             RET
```

et CALL 43010 affiche R.

Exemple 2 : Mettons une chaîne d'expansion, par exemple LOAD", associée au signe d'expansion 141. Au préalable, nous affectons le signe d'expansion 141 à la touche CTRL - Ø. Il s'agit de la touche Ø du petit clavier numérique qui sera enfoncée avec la touche CTRL.

```
43000  4C,4F,41,44,22   DEF LOAD"

43010  3E,0F            LD A,15        :No de la touche 0
43012  06,8D            LD B,141       :signe d'expansion
43014  CD,33,BB         CALL BB33      :affectation
43017  0E,05            LD C,5         :longueur chaine
43019  21,F8,A7         LD HL,43000    :adr. de la chaine
43022  CD,0F,BB         CALL BB0F      ;l'associe a 141
43025  C9               RET
```

Faites CALL 43010 pour mettre en place. Dorénavant, CTRL - Ø affiche LOAD".

LE TEXTE

Cette section manipule les caractères sur l'écran. Elle contrôle huit voies différentes, chacune ayant une fenêtre d'écran associée. Elle traite également certains caractères comme des caractères de contrôle pour amener certaines commandes.

Cette section emploie deux systèmes de coordonnées : logiques et physiques. Celles-ci sont surtout à usage interne. Les deux systèmes utilisent des nombres signés de huit bits et travaillent en position de caractère. Cette position varie selon le mode d'écran. Les colonnes sont numérotées de gauche à droite et les rangées de haut en bas. Les coordonnées logiques (au sens français du terme) numérotent les colonnes de 1 à 40, et les rangées de 1 à 25 et les coordonnées physiques de 0 à 39 et de 0 à 24.

Caractères

- *BB5A :* Envoie un caractère vers l'écran à la position courante. La routine obéit aux codes de contrôle 0 à 31, dont certains comportent plusieurs paramètres qui doivent tous être envoyés. (voir chapitre 9 du guide).
  Entrée : A contient le caractère à envoyer.
  Sortie : Les registres sont préservés.

- *BB5D :* Envoie un caractère sur l'écran à la position courante. Les codes de contrôle ne sont pas traités.
  Entrée : A contient le caractère à envoyer.
  Sortie : Les registres simples sont indéfinis.

- *BB60 :* Lit un caractère de l'écran à la position courante.
  Entrée : Aucune condition.
  Sortie : S'il y a un caractère correct, $C_i=1$ et A contient ce caractère.
  S'il y a un caractère incorrect, $C_i=0$ et A=0.
  Les registres sont préservés.

  Il faut prendre des précautions pour la reconnaissance d'un caractère. Par exemple, le caractère "espace" est reconnu pour autant que vous n'ayez pas changé de couleur papier localement. Nous avons parlé de cela dans le jeu qui termine le chapitre précédent.

- *BB63 :* Rend disponible ou indisponible l'écriture de caractères graphiques. Fonctionne comme un commutateur inverseur.
  Entrée : A différent de 0 pour l'écriture graphique.
  A=0 pour supprimer l'écriture graphique.
  Sortie : A est indéfini.

Fenêtres

- *BB66 :* Fixe les limites de la fenêtre de la voie courante.
  Entrée : H contient l'abscisse physique d'un côté.
  D contient l'abscisse physique de l'autre côté.
  L contient l'ordonnée physique d'un côté.
  E contient l'ordonnée physique de l'autre côté.
  La colonne de gauche est soit H, soit D, le plus petit des deux.
  La rangée du dessus est soit L, soit E, le plus petit des deux.
  Sortie : Les registres simples sont indéfinis.

  On emploie ici les coordonnées physiques. Le curseur est déplacé au-dessus à gauche de la fenêtre. Si les limites données sont trop grandes, la fenêtre est réduite pour s'inscrire dans l'écran.

- *BB69 :* Demande les dimensions de la fenêtre de la voie courante.
  Entrée : Pas de condition.
  Sortie : Si la fenêtre couvre tout l'écran $C_i=0$.
  Si la fenêtre est une partie de l'écran $C_i=1$.
  H contient la colonne la plus à gauche.
  D contient la colonne la plus à droite.
  L contient la rangée la plus haute.
  E contient la rangée la plus basse.
  A est indéfini.
  Les limites sont données en coordonnées physiques.

- *BB6C :* Efface la fenêtre de la voie courante.
  Entrée : Aucune.
  Sortie : Les registres sont indéfinis.
  Le curseur est placé au-dessus à gauche de la fenêtre.

Curseur

- *BB6F :* Déplace la position du curseur horizontalement.
  Entrée : A contient la nouvelle colonne en abscisse logique.
  Sortie : A, H et L sont indéfinis.
  Le curseur peut se déplacer hors de la fenêtre. Il y sera remis par la routine BB87.
  Ceci est également valable pour les trois routines suivantes.

- *BB72 :* Déplace la position du curseur verticalement.
  Entrée : A contient la nouvelle rangée en ordonnée logique.
  Sortie : A, H et L indéfinis.

- *BB75 :* Déplace la position du curseur.
  Entrée : H contient le numéro de colonne en coordonnées logiques.
  L contient le numéro de rangée en coordonnées logiques.
  Sortie : A, H et L indéfinis.

- *BB78 :* Demande la position du curseur.
  Entrée : Aucune condition.
  Sortie : H contient le numéro de colonne logique.
  L contient le numéro de rangée logique.
  A contient le nombre de rotations.
  Le nombre de rotations est décrémenté quand la fenêtre exécute une rotation vers le dessus, et incrémenté par rotation en dessous. Ceci n'est utile que comparé à la valeur précédente pour savoir si la fenêtre a subi une rotation.

- *BB7B :* Permet à l'utilisateur d'afficher le curseur.
  Entrée : Aucune condition.
  Sortie : A indéfini.

□ *BB7E :* Permet à l'utilisateur de supprimer le curseur.
Entrée : Aucune.
Sortie : A indéfini.

□ *BB81 et BB84 :* Sont employées par le système. Elles agissent comme BB7B et BB7E.

□ *BB87 :* Teste la validité de la position du curseur dans une fenêtre.
Entrée : H contient le numéro de colonne en coordonnées logiques.
L contient le numéro de rangée en coordonnées logiques.
Sortie : Si l'affichage à cette position ne provoque pas de rotation : $C_i=1$, B est indéfini.
Si l'affichage provoque une rotation vers le dessus : $C_i=0$, B=FF.
Si l'affichage provoque une rotation vers le bas : $C_i=0$, B=0.
H contient la colonne où le caractère sera affiché.
L contient la rangée où le caractère sera affiché.
A est indéfini.

Encres

□ *BB90 :* Met l'encre de la plume pour la voie courante.
Entrée : A contient l'encre.
Sortie : A, H et L indéfinis.
L'encre est masquée suivant le mode d'écran pour la rendre correcte si elle ne l'est pas.

□ *BB93 :* Demande l'encre de la plume utilisée par la voie courante.
Entrée : Aucune.
Sortie : A contient l'encre de la plume.
Les registres sont préservés.

□ *BB96 :* Met l'encre du papier pour la voie courante.
Entrée : A contient l'encre.
Sortie : A, H et L indéfinis.
L'encre est masquée suivant le mode d'écran.

□ *BB99 :* Demande l'encre du papier.
Entrée : Aucune.
Sortie : A contient l'encre du papier.
Les registres sont préservés.

□ *BB9C :* Echange les encres plume et papier.
Entrée : Aucune.
Sortie : A, H et L sont indéfinis.

□ *BB9F :* Met le mode d'écriture opaque ou transparent pour le fond du caractère.
Entrée : A=0 pour le mode opaque.
A≠0 pour le mode transparent.
Sortie : A, H et L indéfinis.

□ *BBA2 :* Demande le mode d'écriture du fond.
Entrée : Aucune.
Sortie : A=0 pour le mode opaque.
A≠0 pour le mode transparent.
D, E, H et L indéfinis.

Matrices

□ *BBA5 :* Demande l'adresse de la matrice d'un caractère.
Entrée : A contient le caractère.
Sortie : Si la matrice a été définie par l'usager : C_i=1.
Si la matrice est dans la ROM inf. : C_i=0.
HL contient l'adresse.
A est indéfini.

La matrice d'un caractère se compose de huit octets successifs ; le premier définit la ligne supérieure du caractère, son bit 7 est le plus à gauche, et ainsi de suite.

□ *BBA8 :* Met une matrice de caractère définie par l'usager.
Entrée : A contient le caractère à redéfinir.
HL contient l'adresse de la matrice.
Si A ne contient pas un caractère défini par l'usager, rien ne se passe.
Sortie : C_i=1 si le caractère est défini par l'usager.
C_i=0 si le caractère n'est pas défini par l'usager.
Les registres sont indéfinis.

□ *BBAB :* Met une table de matrices définies par l'usager ainsi que le nombre de caractères dans cette table. La table est initialisée avec les matrices courantes.
Entrée : DE contient le premier caractère dans la table (0-255).
HL contient l'adresse de départ de la nouvelle table.
Sortie : S'il n'y avait pas de table définie par l'usager : C_i=0, A et HL indéfinis.
S'il y avait déjà une table : C_i=1, A contient le premier caractère de la vieille table.
HL contient l'adresse de la vieille table.
B, C, D et E sont indéfinis.

□ *BBAE :* Demande l'adresse de la table des matrices définies par l'usager ainsi que son premier caractère.
Entrée : Aucune.
Sortie : S'il n'y a aucune table définie par l'usager : C_i=0, A et HL indéfinis.
S'il y en a une : C_i=1, A contient le premier caractère dans la table.
HL contient l'adresse de départ de la table.
Les autres registres sont préservés.

Codes de contrôle

□ *BBB1 :* Demande l'adresse de la table des codes de contrôle.
Entrée : Aucune.

Sortie : HL contient l'adresse.
Les autres registres sont préservés.
L'exemple 3 ci-dessous, montre la construction de la table.

Voies

□ *BBB4 :* Sélectionne une voie.
Entrée : A contient la voie demandée.
A sera masqué avec 07 pour le rendre correct.
Sortie : A contient la voie sélectionnée précédemment.
H et L sont indéfinis.

□ *BBB7 :* Echange l'état de deux voies. Les numéros des voies ne changent pas.
Entrée : B contient un numéro de voie (masquée avec 07).
C contient l'autre numéro de voie (masquée avec 07).
Sortie : Les registres simples sont indéfinis.
Les attributs échangés sont : l'encre plume, l'encre papier, la position du curseur, les limites de la fenêtre, le compteur de rotations de la fenêtre, le curseur disponible ou non, le mode d'écriture.

Exemple 3 : Extrayons l'adresse de la table des codes de contrôle.

```
43010  CD.B1,BB          CALL BBB1        ;adresse table
43013  22.F8.A7          LD (43000),HL    ;la stocke
43016  C9                RET
```

Faites CALL 43010 puis PRINT PEEK(43000)+256*PEEK(43001) qui affiche 45763.

La table des codes de contrôle s'étend de B2C3-45763 à B322-45858. Cette table contient 96 C.M. ou 32 groupes de 3 C.M. Pour chacun des 32 codes de contrôle, le premier C.M. donne le nombre de paramètres et les deux suivants l'adresse de la routine d'exécution.
Les codes de contrôle sont décrits dans votre guide.
Le numéro du code, suivi des paramètres éventuels, est envoyé avec la routine BB5A. Nous avons déjà vu l'emploi de ceci pour afficher des messages dans le jeu du chapitre précédent.

Pour faire sonner la clochette, faites :

```
43017  3E.07             LD A.7           :code de controle 7
43019  CD.5A.BB          CALL BB5A        :envoie le carac.
43022  C9                RET
```

Faites CALL 43017 pour faire sonner la clochette.

Exemple 4 : A l'initialisation, la ROM BASIC établit une table de matrices de caractères disponibles pour l'usager. Extrayons l'adresse de cette table :

```
43010  CD.AE.BB        CALL BBAE
43013  D0              RET NC            :ret. si table abs.
43014  22.F8.A7        LD (43000).HL     :adresse table
43017  32,FA,A7        LD (43002).A      :1ier caractere
43020  C9              RET
```

Faites CALL 43010 puis

PRINT PEEK(43000)+256−PEEK(43001)

qui affiche 43904 puis

PRINT PEEK(43002)

qui affiche 240.

A AB80-43904 commence une table de seize matrices de caractères de codes 240 à 255. Cette table s'étend jusqu'à ABFF−44031. Elle contient les seize matrices de huit octets chacune, qui représentent les caractères de codes 240 à 255 qui y ont été transférés à l'initialisation.

Exemple 5 : Remplaçons le caractère 240 par un petit voilier :

```
43000  20.30,38.3C     DEF matrice du
43004  3E.FF.7E.00     DEF voilier

43010  3E.F0           LD A.240d         :code caractere
43012  21.F8.A7        LD HL.43000       :adresse matrice
43015  CD.A8.BB        CALL BBA8         :mise en place
43018  C9              RET
```

Faites CALL 43010 puis

PRINT CHR$(240)

pour afficher le voilier.

Nous avons dessiné le voilier comme suit :

0 0 1 0 0 0 0 0	20
0 0 1 1 0 0 0 0	30
0 0 1 1 1 0 0 0	38
0 0 1 1 1 1 0 0	3C
0 0 1 1 1 1 1 0	3E
1 1 1 1 1 1 1 1	FF
0 1 1 1 1 1 1 0	7E
0 0 0 0 0 0 0 0	00

Ces octets sont mis dans cet ordre à partir de 43000, puisque nous chargeons HL avec cette valeur.

Remarque importante : Faites la commande BASIC :

SYMBOL AFTER 32

et vous avez le message d'erreur "improper argument".
Que se passe-t-il ?

Le BASIC nous réserve seize caractères à redéfinir dont les matrices occupent les adresses 43904 et suivantes. Avec la commande ci-dessus, le système essaye de nous réserver de la place pour les matrices de 224 caractères de codes 32 à 255. Or, nous avons réduit HIMEM à la ligne 10010 de notre programme de chargement de la fin de la première partie. Le système ne peut satisfaire notre demande.

Si vous voulez redéfinir tous les caractères (ou, du moins plus de seize caractères), il faut le faire avec HIMEM=43903. Ensuite, vous devez modifier le programme de chargement avant de le faire tourner, pour charger vos C.M. plus bas. Pour 224 matrices (SYMBOL AFTER 32), le système réserve une zone de 42240 à 44031 soit 1792 octets. Si vous souhaitez charger vos C.M. à partir de 41000, faites :

10010 MEMORY 40999:a=HIMEM+1
10060 changez 43903 en 42239
10240 changez 43902 en 42238

Exemple 6 : Montrons les caractères affichables agrandis huit fois linéairement. L'instruction SYMBOL AFTER recopie les matrices des caractères désignés, de la ROM inférieure dans la mémoire RAM.
Faites la commande CTRL-SHIFT-ESC, puis entrez le BASIC :

```
10 MODE 1:SYMBOL AFTER 32
20 FOR n=42240 TO 44031
30 a$=BIN$(PEEK(n),8)
```

```
40 FOR b=1 TO 8
50 IF MID$(a$,b,1)="1" THEN MID$(a$,b,1)
=" ":REM pour remplacer 1 par espace
60 NEXT b
70 PRINT 32+FIX((n-42240)/8);TAB(16)a$:c
=c+1
80 IF c=8 THEN c=0:PRINT TAB(16)STRING$(
8,"-"):REM pour separer
90 NEXT n
```

RUN

Si vous voulez examiner une matrice particulière, faites ESC, puis poussez sur une autre touche.

LE GRAPHISME | 7

Nous terminons ce livre en continuant la description des routines accessibles par le bloc de sauts principal, en les agrémentant d'exemples.

Une des possibilités du système Amstrad est de pouvoir étendre le nombre de commande BASIC. Les RSX, extensions du système résident, procurent cette possibilité. Nous en donnons un exemple pour afficher des cercles et des ellipses.

Nous examinons ensuite la manière de procéder pour déplacer sur l'écran des objets multicolores. Vous pourrez étendre ce procédé pour déplacer des lutins.

1

Le bloc de sauts principal-suite

LE GRAPHISME

Cette section permet de mettre des points sur l'écran ou de les tester, et de tracer des lignes.

Pour les besoins du BASIC, l'écran est considéré comme un écran idéal de 640 points de large sur 400 points de haut, quel que soit le mode d'écran. Ainsi la commande PLOT 320,200 affiche un point central, quel que soit le mode employé.

Nous savons qu'en C.M. il y a 200 lignes de points en hauteur et que le nombre de points par ligne varie selon le mode.

La section graphisme emploie quatre systèmes de coordonnées. L'usager utilise les coordonnées utilisateur ou les coordonnées relatives, quelquefois les coordonnées standards. Ces trois systèmes sont très proches. En coordonnées standards, l'origine est en dessous à gauche. En coordonnées utilisateur, l'origine est donnée par l'usager. Enfin, en coordonnées relatives, l'origine est la position courante : la dernière utilisée. Ces trois systèmes emploient des nombres de seize bits avec signe.

Les coordonnées de base, 4e système, utilisées par le système ont leur origine dans le coin gauche inférieur et travaillent en points pixels dont le nombre dépend du mode d'écran. Ceci est évident puisque notre ordinateur fonctionne en C.M. Les coordonnées sont exprimées en nombre de seize bits sans signe.

Position courante

- *BBC0 :* Déplace la position courante à une position absolue.
  Entrée : DE contient l'abscisse x.
  HL contient l'ordonnée y.
  Sortie : Les registres sont indéfinis.
  La nouvelle position est en coordonnées usager, c'est-à-dire relatives à l'origine usager.

- *BBC3 :* Déplace la position courante relativement à elle-même.
  Entrée : DE contient l'offset de l'abscisse x, avec signe.
  HL contient l'offset de l'ordonnée y, avec signe.
  Sortie : Registres indéfinis.
  Ces deux routines peuvent déplacer la position courante hors de l'écran.

- *BBC6 :* Demande où se trouve la position courante.
  Entrée : Aucune.
  Sortie : DE contient l'adresse x.
  HL contient l'ordonnée y.
  A est indéfini.
  La position courante est donnée en coordonnées usager.

- *BBC9 :* Met l'origine des coordonnées usager.
  Entrée : DE contient l'abscisse x de l'origine.
  HL contient l'ordonnée y de l'origine.
  Sortie : Les registres sont indéfinis.
  L'origine est donnée en coordonnées standards où (0,0) est l'origine par défaut.

- *BBCC :* Demande l'origine.
  Entrée : Aucune.
  Sortie : DE contient l'abscisse x de l'origine.
  HL contient l'ordonnée y de l'origine.
  Les autres registres sont préservés.
  La position de l'origine est donnée en coordonnées standards.

Fenêtre

- *BBCF :* Met les positions horizontales de la fenêtre.
  Entrée : DE contient l'abscisse standard x d'un côté.
  HL contient l'abscisse standard x de l'autre côté.
  Le côté gauche sera le plus petit des deux.
  Sortie : Les registres sont indéfinis.
  La fenêtre est tronquée pour que ses dimensions contiennent un nombre entier d'octets, selon le mode d'écran.

□ *BBD2 :* Met les positions verticales de la fenêtre graphique.
Entrée : DE contient l'ordonnée standard y d'un côté
HL contient l'ordonnée standard y de l'autre côté.
Le côté supérieur sera le plus grand des deux.
Sortie : Registres indéfinis.
La fenêtre est tronquée comme pour la routine précédente.

□ *BBD5 :* Demande la position horizontale de la fenêtre graphique.
Entrée : Aucune.
Sortie : DE contient l'abscisse standard x du côté gauche.
HL contient l'abscisse standard x du côté droit.
A est indéfini.

□ *BBD8 :* Demande la position verticale de la fenêtre graphique.
Entrée : Aucune.
Sortie : DE contient l'ordonnée standard y du côté supérieur.
HL contient l'ordonnée standard y du côté inférieur.
A est indéfini.

□ *BBDB :* Eclaircit la fenêtre graphique.
Entrée : Aucune.
Sortie : Les registres sont indéfinis.
La position graphique est déplacée à l'origine des coordonnées usager.

Encres

□ *BBDE :* Met l'encre de plume.
Entrée : A contient l'encre demandée.
Sortie : A est indéfini.
L'encre est masquée, selon le mode d'écran en usage, pour la réduire à 16, 4 ou 2.

□ *BBE1 :* Demande l'encre de plume.
Entrée : Aucune.
Sortie : A contient l'encre de plume.
Les registres sont préservés.

□ *BBE4 :* Met l'encre du papier graphique.

□ *BBE7 :* Demande l'encre du papier graphique.
Fonctionnent comme les deux routines précédentes.

Affichage et test d'un point

□ *BBEA :* Affiche un point à une position absolue.
Entrée : DE contient l'abscisse x du point à afficher.
HL contient l'ordonnée y du point à afficher.

Sortie : Registres indéfinis.
La position est donnée en coordonnées usager, c'est-à-dire relatives à l'origine usager.

□ *BBED :* Affiche un point relativement à la position courante.
Entrée : DE contient l'offset avec signe de x.
HL contient l'offset avec signe de y.
Sortie : Registres indéfinis.
La position est en coordonnées relatives.

□ *BBF0 :* Teste un point à une position absolue.
Entrée : DE contient l'abscisse usager x à tester.
HL contient l'ordonnée usager y à tester.
Sortie : A contient l'encre du point spécifié.
Registres indéfinis.
Position en coordonnées usager.

□ *BBF3 :* Teste un point relativement à la position courante.
Entrée : DE contient un offset avec signe x.
HL contient un offset avec signe y.
Sortie : A contient l'encre du point spécifié.
Registres indéfinis.
Position en coordonnées usager.

Tracé d'une ligne

□ *BBF6 :* Trace une ligne vers une position absolue à partir de la position précédente.
Entrée : DE contient l'abscisse usager x du point final.
HL contient l'ordonnée usager y du point final.
Sortie : Registres indéfinis.
Position finale en coordonnées usager.

□ *BBF9 :* Trace une ligne relativement à la position courante.
Entrée : DE contient l'offset avec signe x du point final.
HL contient l'offset avec signe y du point final.
Sortie : Registres indéfinis.
Position finale en coordonnées usager.

Affichage d'un caractère

□ *BBFC :* Affiche un caractère à la position graphique courante.
Entrée : A contient le caractère.
Sortie : Registres indéfinis.
La position graphique courante est le point supérieur gauche du caractère. Cette position est ensuite déplacée d'un caractère vers la droite.

Exemple 1 : Dessinons trois carrés sur l'écran, en exploitant les routines vues ci-dessus :

```
43000  3E,01                 LD A,1
43002  CD,DE,BB              CALL BBDE      ;encre 1 de plume
43005  11,AA,00              LD DE,170      ;x
43008  21,AA,00              LD HL,170      ;y
43011  CD,C0,BB              CALL BBC0      ;position absolue
43014  CD,28,A8              CALL ROUT      ;43048 aff. 1 carre
43017  3E,02                 LD A,2         ;
43019  CD,DE,BB              CALL BBDE      ;encre 2 de plume
43022  11,E6,00              LD DE,230      ;x
43025  21,6E,00              LD HL,110      ;y
43028  CD,C0,BB              CALL BBC0      ;position absolue
43031  CD,28,A8              CALL ROUT      ;dessine 1 carre
43034  3E,03                 LD A,3
43036  CD,DE,BB              CALL BBDE      ;encre 3 de plume
43039  11,22,01              LD DE,290
43042  21,32,00              LD HL,50
43045  CD,C0,BB              CALL BBC0
43048  11,B4,00      ROUT    LD DE,180
43051  21,00,00              LD HL,0
43054  CD,F9,BB              CALL BBF9      ;ligne relative
43057  11,00,00              LD DE,0
43060  21,B4,00              LD HL,180
43063  CD,F9,BB              CALL BBF9      ;ligne relative
43066  11,4C,FF              LD DE,-180     ;nombre avec signe
43069  21,00,00              LD HL,0
43072  CD,F9,BB              CALL BBF9      ;ligne relative
43075  11,00,00              LD DE,0
43078  21,4C,FF              LD HL,-180
43081  CD,F9,BB              CALL BBF9
43084  C9                    RET
```

Faites la commande :

MODE 1:CALL 43000

l'écran affiche 3 carrés en 3 couleurs.

La routine fonctionne dans les trois modes. En mode 0, les points contiennent plus de pixels et en mode 2, moins. Dans ce dernier mode, les couleurs possibles sont jaune et bleue (invisible).

Exemple 2 : Affichons un texte en mode graphique, plusieurs fois, en faisant varier les coordonnées graphiques :

```
43000  53,41,4C,55,54        DEF SALUT
43005  FF                    DEF fin de message
```

```
43010  DD,21,F8,A7   AFF    LD IX,43000
43014  DD,7E,00            ┌→LD A,(IX+0)
43017  FE,FF               │ CP FF
43019  C8                  │ RET Z
43020  CD,FC,BB            │ CALL BBFC        ;affiche
43023  DD,23               │ INC IX
43025  18,F3               └─JR -13+2
43027  11,40,01      ENT     LD DE,320
43030  21,40,01              LD HL,320
43033  06,14                 LD B,20
43035  C5                  ┌→PUSH BC
43036  E5                  │ PUSH HL
43037  D5                  │ PUSH DE
43038  CD,C0,BB            │ CALL BBC0        ;pos. absolue
43041  CD,02,A8            │ CALL AFF         :43010
43044  D1                  │ POP DE
43045  E1                  │ POP HL
43046  C1                  │ POP BC
43047  2B                  │ DEC HL           ;ordonnee
43048  2B                  │ DEC HL           :  moins 2
43049  13                  │ INC DE           ;abscisse
43050  13                  │ INC DE           :  plus 2
43051  10,EE               └─DJNZ -18+2
43053  C9                    RET
```

Faites la commande :

CLS:CALL 43027 voyez la glissade.

Cette routine fonctionne dans les trois modes d'écran.
Nous employons la sous-routine AFF pour afficher le texte avec IX comme pointeur de caractères. La routine principale ENT manipule la position absolue à chaque passage de la boucle contrôlée par B. Le mode graphique ne permet pas de travailler en transparent comme le permet le mode texte, pour afficher du texte sur des graphiques, en coordonnées caractères.

L'ÉCRAN

La section écran est utilisée par les sections texte et graphisme. Elle agit sur les caractéristiques d'écran communes à ces deux sections : mode d'écran, couleurs, position, etc.
Dans cette section, la position d'affichage est calculée en adresses d'écran. On accède aux octets d'écran selon le plan de la mémoire d'écran que nous avons vu précédemment aux parties 3 et 4.

□ *BC05 :* Met l'offset du premier caractère d'écran.
Entrée : HL contient l'offset demandé.
Sortie : A, H et L indéfinis.
L'offset est masqué par 7FE pour qu'il puisse agir sur tout l'écran, et pour le rendre pair, car l'écran ne peut se déplacer que par multiple de deux octets.

□ *BC0B :* Demande l'offset d'écran.
Entrée : Aucune.
Sortie : HL contient l'offset courant.
Les registres sont préservés.

Mode

□ *BC0E :* Met le mode d'écran.
Entrée : A contient le mode demandé, donc 0, 1 ou 2.
Sortie : Registres indéfinis.
Mode 0 : 160 × 200 points, 20 × 25 caractères.
Mode 1 : 320 × 200 points, 40 × 25 caractères.
Mode 2 : 640 × 200 points, 80 × 25 caractères.

□ *BC11 :* Demande le mode d'écran.
Entrée : Aucune condition.
Sortie : Si le mode est 0 : C_i=1, Z=0, A=0.
Si le mode est 1 : C_i=0, Z=1, A=1.
Si le mode est 2 : C_i=0, Z=0, A=2.
Les registres sont préservés.

□ *BC14 :* Eclaircit toute la mémoire d'écran.
Entrée : Aucune.
Sortie : Registres indéfinis.
L'offset d'écran est mis à 0.

Adresses d'écran

□ *BC1A :* Calcule l'adresse d'écran d'un caractère (point supérieur gauche de sa matrice).
Entrée : H contient la colonne du caractère en coordonnées physiques.
L contient la rangée du caractère en coordonnées physiques.
(La position (0,0) est au-dessus à gauche).
Sortie : HL contient l'adresse d'écran du point supérieur gauche du caractère.
B contient la largeur en octets d'un caractère.
A est indéfini.

□ *BC1D :* Calcule l'adresse d'écran d'un point donné en coordonnées de base.
Entrée : DE contient l'abscisse de base x
HL contient l'ordonnée de base y.
Sortie : HL contient l'adresse d'écran du point.

C contient le masque pour ce point.
B contient un de moins que le nombre de points dans 1 octet.
A, D et E sont indéfinis.

Le nombre de points par octet est deux en mode 0, quatre en mode 1, huit en mode 2. Cette routine et la précédente ne font pas de test de validité des entrées. Par conséquent l'adresse d'écran peut ne pas être valable.

□ *BC20 :* Calcule l'adresse d'écran de l'octet à droite de l'adresse d'écran fournie.
Entrée : HL contient une adresse d'écran.
Sortie : HL contient une adresse d'écran mise à jour.
A est indéfini.

□ *BC23 :* Calcule l'adresse d'écran de l'octet à gauche de l'adresse fournie.
Entrée : HL contient une adresse d'écran.
Sortie : HL contient une adresse d'écran mise à jour.
A est indéfini.

□ *BC26 :* Calcule l'adresse d'écran de l'octet en dessous de l'adresse fournie.
Entrée : HL contient une adresse d'écran.
Sortie : HL contient une adresse d'écran mise à jour.
A est indéfini.

□ *BC29 :* Calcule l'adresse d'écran de l'octet au-dessus de l'adresse fournie.
Entrée : HL contient une adresse d'écran.
Sortie : HL contient une adresse d'écran mise à jour.
A est indéfini.

Aucun test de validité n'est fait pour ces 4 routines.

Encres

□ *BC2C :* Convertit une encre dans sa forme codée pour agir sur les points d'un octet suivant le mode.
Entrée : A contient un numéro d'encre.
Sortie : A contient l'encre codée.
Les registres sont préservés.

Ceci agit différemment selon les modes :

Point	mode 0	mode 1	mode 2
le plus à gauche	bits 1,5,3,7	bits 3,7	bit 7
			bit 6
		bits 2,6	bit 5
			bit 4
	bits 0,4,2,6	bits 1,5	bit 3
			bit 2
		bits 0,4	bit 1
			bit 0

puisqu'il y a deux, quatre ou huit points par octet.

- *BC2F :* Convertit une encre codée en son numéro d'encre.
  Entrée : A contient l'encre codée.
  Sortie : A contient le numéro d'encre.
  Les registres sont préservés.

- *BC32 :* Met deux couleurs à une encre.
  Entrée : A contient le numéro d'encre.
  B contient la première couleur.
  C contient la deuxième couleur.
  Si B et C sont différents, il y a clignotement des deux couleurs.
  Sortie : Les registres sont indéfinis.

- *BC35 :* Demande les couleurs d'une encre.
  Entrée : A contient le numéro d'encre.
  Sortie : B contient la première couleur.
  C contient la deuxième couleur.
  Les autres registres simples sont indéfinis, y compris A.

- *BC38 :* Met deux couleurs pour la bordure.
  Entrée : B contient la première couleur.
  C contient la deuxième couleur.
  Sortie : Registres indéfinis.
  B et C différents font clignoter la bordure.

- *BC3B :* Demande les couleurs de la bordure.
  Entrée : Aucune.
  Sortie : B contient la première couleur.
  C contient la deuxième couleur.
  Les autres registres sont indéfinis.

- *BC3E :* Met la période de clignotement pour toutes les encres et la bordure.
  Entrée : H contient la période de la première couleur.
  L contient la période de la deuxième couleur.
  Sortie : A est indéfini, de même que H et L.
  Les périodes sont données en 1/50 de seconde. La mise par défaut est 10 (soit 1/5 seconde), pour les deux couleurs.

- *BC41 :* Demande les périodes de clignotement.
  Entrée : Aucune.
  Sortie : H contient la période de la première couleur.
  L contient la période de la deuxième couleur.
  A est indéfini.
  Les périodes sont en 1/50 seconde.

Divers

- *BC44 :* Remplit une zone de caractères d'écran avec une encre.
  Entrée : A contient l'encre codée.
  H contient la colonne gauche de la zone.
  D contient la colonne droite de la zone.
  L contient la rangée supérieure de la zone.
  E contient la rangée inférieure de la zone.
  Sortie : Les registres sont indéfinis.
  La zone est délimitée en coordonnées physiques de caractères.
  Aucun test de validité n'est fait.

- *BC47 :* Remplit une zone d'octets d'écran avec une encre.
  Entrée : C contient l'encre codée.
  HL contient l'adresse d'écran du coin gauche supérieur de la zone.
  D contient la largeur de la zone en octets, sans signe.
  E contient la hauteur de la zone en lignes d'écran sans signe.
  Sortie : Registres indéfinis.
  Aucun test de validité n'est fait.

- *BC4A :* Echange les couleurs d'un caractère.
  Entrée : B contient une encre codée.
  C contient une autre encre codée.
  H contient le numéro de colonne.
  L contient le numéro de rangée.
  Sortie : Registres indéfinis.
  La position du caractère est donnée en coordonnées physiques.
  Aucun test de validité.

- *BC4D :* Déplace l'écran de huit lignes (1 caractère).
  Entrée : Déplacement vers le bas : B=0.
  Déplacement vers le haut : B≠0.
  A contient l'encre codée pour éclaircir la nouvelle rangée.
  Sortie : Registres indéfinis.

- *BC50 :* Déplace une partie de l'écran de huit lignes (1 caractère).
  Entrée : Déplacement vers le bas : B=0.
  Déplacement vers le haut : B≠0.
  A contient l'encre codée.
  H contient la colonne gauche de la zone.
  D contient la colonne droite de la zone.
  L contient la rangée supérieure de la zone.
  E contient la rangée inférieure de zone.
  Sortie : Registres indéfinis.
  On emploie les coordonnées physiques des caractères.Aucun test de validité.

□ *BC53 :* Convertit une matrice de caractères en un jeu de masques de points suivant le mode.
Entrée : HL contient l'adresse de la matrice.
DE contient l'adresse d'une zone pour le jeu de masques.
Sortie : Registres indéfinis.
La série de masques couvre le nombre d'octets nécessaires pour le caractère. Par conséquent, chaque octet de la matrice est converti en quatre octets pour le mode 0, deux pour le mode 1 et un pour le mode 2. DE devra pointer sur une zone libre de trente-deux, seize ou huit octets.

□ *BC56 :* Convertit un caractère sur l'écran en sa matrice.
Entrée : A contient l'encre codée pour l'assortir au caractère.
H contient la colonne physique du caractère.
L contient la rangée physique du caractère.
DE contient l'adresse d'une zone pour y mettre la matrice de huit octets.
Sortie : Registres indéfinis.
Aucun test de validité.

□ *BC59 :* Met le mode d'écriture graphique.
Entrée : A contient le mode d'écriture demandé.
Sortie : Registres indéfinis.
Les modes d'écriture sont :
0 mode forcé : nouvelle encre ;
1 mode XOR : nouvelle XOR ancienne ;
2 mode AND : nouvelle AND ancienne ;
3 mode OR : nouvelle OR ancienne.

□ *BC5C :* Affiche un point sur l'écran à la position donnée pour une adresse d'écran et un masque de points.
Entrée : B contient l'encre codée à employer.
C contient le masque pour les points.
HL contient l'adresse d'écran.
Sortie : A est indéfini.
Le point sera affiché, quel que soit le mode d'écriture graphique.
Aucun test de validité.

□ *BC5F :* Trace une ligne horizontale.
Entrée : A contient l'encre codée.
DE contient l'abscisse de base x du départ de la ligne.
BC contient l'abscisse de base x de la fin de la ligne.
HL contient l'ordonnée de base y de la ligne.
Sortie : Registres indéfinis.

□ *BC62 :* Trace une ligne verticale.
Entrée : A contient l'encre codée.
DE contient l'abscisse de base x de la ligne.

HL contient l'ordonnée de base y du début de ligne.
BC contient l'ordonnée de base y de fin de ligne.
Sortie : Registre indéfinis.

Exemple 3 : Employons la mise de l'offset d'écran, pour faire bouger un texte.
Entrez les C.M. suivants :

```
43010  CD,0B,BC          CALL BC0B          ;demande l'offset
43013  22,F8,A7          LD (43000),HL      ;la stocke
43016  C9                RET
43017  2A,FA,A7          LD HL,(43002)      ;offset mis a jour
43020  CD,05,BC          CALL BC05          ;met l'offset
43023  C9                RET
```

Entrez ensuite le BASIC :

```
10 CLS:LOCATE 15,12:PRINT"codez"
20 CALL 43010
30 a=PEEK(43000)+256*PEEK(43001)
40 FOR n=1 TO 10
50 a=a+80
60 GOSUB 150
70 a=a-10
80 GOSUB 150
90 a=a-80
100 GOSUB 150
110 a=a+10
120 GOSUB 150
130 NEXT n
140 END
150 IF a<0 THEN a=a+2048
160 h=FIX(a/256):l=a-256*h
170 POKE 43002,l:POKE 43003,h
180 CALL 43017
190 FOR t=1 TO 300:NEXT
200 RETURN
```

et voyez la ronde du message “codez”.
Nous avons vu au chapitre 3 que l'écran se divisait en huit blocs de 2 K. d'octets chacun. C'est pour cette raison que l'on additionne 2048 dans la ligne 150. L'offset donné est d'ailleurs masqué par 7FE - 2046d, pour le rendre pair et situé dans un bloc d'écran.

Exemple 4 : Affichons des couleurs en employant des routines d'écran :

```
43010  3E,00                 LD A,0
43012  CD,0E,BC              CALL BC0E      ;mode 0
43015  11,0A,00              LD DE,10       ;abscisse 10
43018  21,40,00              LD HL,64       ;ordonnee 64
43021  3E,0F                 LD A,15        ;encre a 15
43023  D5                    PUSH DE
43024  E5                    PUSH HL
43025  F5                    PUSH AF
43026  CD,1D,BC              CALL BC1D      ;adresse d'ecran
43029  F1                    POP AF         ;No d'encre
43030  F5                    PUSH AF
43031  CD,2C,BC              CALL BC2C      ;encre codee
43034  4F                    LD C,A
43035  16,40                 LD D,64        ;largeur 64 octets
43037  1E,02                 LD E,2         ;hauteur 2 lignes
43039  CD,47,BC              CALL BC47      ;remplit
43042  F1                    POP AF
43043  E1                    POP HL
43044  D1                    POP DE
43045  23                    INC HL         ;ordonnee
43046  23                    INC HL         ;  plus 2
43047  3D                    DEC A          ;encre -1
43048  20,E5                 JR NZ,-27+2    ;boucle 15 fois
43050  C9                    RET
43051  CD,02,A8      ENT     CALL 43010     ;aff. 15x2 lignes
43054  3E,0F                 LD A,15
43056  CD,0F,A8              CALL 43023     ;aff. 15x2 lignes
43059  3E,0F                 LD A,15
43061  CD,0F,A8              CALL 43023     ;aff. 15x2 lignes
43064  C9                    RET
```

faites la commande CALL 43051 et voyez l'affichage multicolore.

Exemple 5 : Dessinons des rectangles multicolores.
Nous employons les routines BC5F et BC62 pour tracer les lignes horizontales et verticales. Notez que le système emploie ces routines pour tracer des lignes obliques. Celles-ci sont divisées en petits segments horizontaux ou verticaux selon la comparaison entre la différence des abscisses et celle des ordonnées.

Pour ralentir le dessin des rectangles, nous employons un programme BASIC pour déterminer au hasard les dimensions des rectangles. Le programme en C.M. se charge du dessin de ces rectangles. Entrez les C.M. suivants :

```
43010  C5            ROUT    PUSH BC
43011  D5                    PUSH DE
43012  E5                    PUSH HL
```

```
43013  F5                        PUSH AF
43014  D4,5F,BC                  CALL NC,BC5F     ;horizontale
43017  DC,62,BC                  CALL C,BC62      ;verticale
43020  F1                        POP AF
43021  E1                        POP HL
43022  D1                        POP DE
43023  C1                        POP BC
43024  C9                        RET

43040  3A,F8,A7        ENT       LD A,(43000)
43043  CD,2C,BC                  CALL BC2C        ;encre codee
43046  ED,5B,FA,A7               LD DE,(43002)    ;abscisse
43050  ED,4B,FE,A7               LD BC,(43006)    ;   "     fin
43054  2A,00,A8                  LD HL,(43008)    ;ordonnee h.
43057  A7                        AND A            ;Ci=0
43058  CD,02,A8                  CALL ROUT        ;43010
43061  2A,FC,A7                  LD HL,(43004)    ;ordonnee 2ieme h.
43064  CD,02,A8                  CALL ROUT
43067  ED,4B,00,A8               LD BC,(43008)    ;ordonnee v.
43071  3F                        CCF              ;Ci=1
43072  CD,02,A8                  CALL ROUT
43075  ED,5B,FE,A7               LD DE,(43006)    ;abscisse 2ime v.
43079  CD,02,A8                  CALL ROUT
43082  C9                        RET
```

L'entrée de la routine est à 43040. On appelle quatre fois la sous-routine ROUT qui affiche une ligne ; deux fois avec C_i=0 pour les deux lignes horizontales et deux fois avec C_i=1 pour les deux verticales. Avant chaque appel, on aménage les paramètres.

Voici le BASIC :

```
10 MODE 0:FOR n=1 TO 20
20 A=FIX(RND*14):REM 0 a 13
30 POKE 43000,A
40 DE=FIX(RND*80):REM 0 a 79
50 POKE 43002,DE:POKE 43003,0
60 HL=FIX(RND*100):REM 0 a 99
70 POKE 43004,HL:POKE 43005,0
80 BCH=159-DE:REM 80 a 159
90 POKE 43006,BCH:POKE 43007,0
100 BCV=199-HL:REM 100 a 199
110 POKE 43008,BCV:POKE 43009,0
120 CALL 43040
130 NEXT
140 FOR b=1 TO 1000:NEXT
150 GOTO 10
9999 END
```

Faites RUN et voyez vingt rectangles s'afficher. Un temps d'attente : puis on recommence. En ligne 20, nous avons mis A pour éviter les encres clignotantes ; vous pouvez mettre 16 au lieu de 14.
Pour les deux routines système, DE est une abscisse et HL une ordonnée. Par contre, BCH est une abscisse pour les lignes horizontales et BCV est une ordonnée pour les verticales. On ne choisit que DE et HL au hasard. BCH et BCV sont calculés pour que chaque rectangle soit symétrique par rapport au centre.

2

Nouvelles commandes BASIC

Le KERNEL possède également une série de routines dans le bloc de sauts principal. La plupart de ces routines traitent des interruptions, des événements, des ROM supérieures additionnelles, etc., qui sortent du cadre de cet ouvrage. Il y a cependant une routine qui permet d'ajouter de nouvelles commandes BASIC au système : c'est une extension du système, ou RSX.

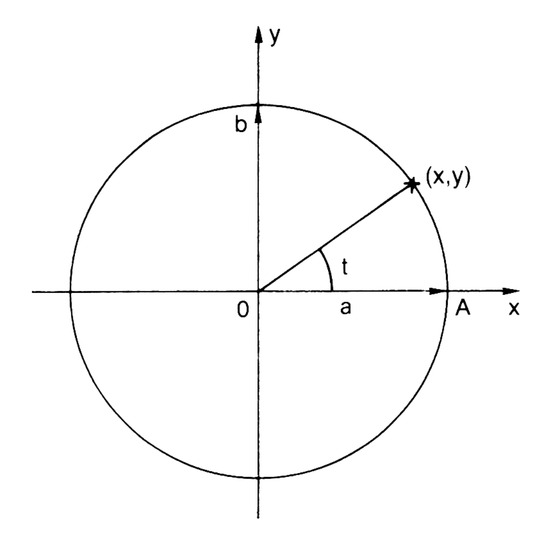
Construisons un programme pour afficher des cercles et, ou, des ellipses sur l'écran. Ce programme en C.M. s'exploite de la manière habituelle. Nous verrons ensuite comment intégrer ce programme dans une nouvelle commande BASIC.

CERCLES ET ELLIPSES

Pour dessiner un cercle, nous employons les formules :

$x = a. \cos t$
$y = b. \sin t$

qui donnent les coordonnées (x,y) d'un point du cercle quand $a=b$ ou d'une ellipse quand $a \neq b$.
Chaque couple de coordonnées est joint au couple suivant par une ligne.



Plus le nombre de points calculés est grand, plus la succession de lignes ressemblera à un cercle ou une ellipse. Nous nous limitons à calculer 9 points par cadran, soit 36 points en tout. Nous calculons donc de 10 en 10 degrés.

Les coordonnées sont calculées par rapport au centre O. Nous devons d'abord fixer les coordonnées usager du centre, ensuite fixer la position de départ que nous avons choisie en A. Enfin, nous calculerons les coordonnées des points et nous les joindrons par une ligne que nous avons choisie dans le sens trigonométrique.

Pour calculer les coordonnées, nous n'avons pas employé les routines BASIC qui prennent du temps et qui nécessitent la virgule flottante, mais les deux premiers chiffres significatifs de la table des sinus et des cosinus.

Entrez les C.M. suivants (paramètres):

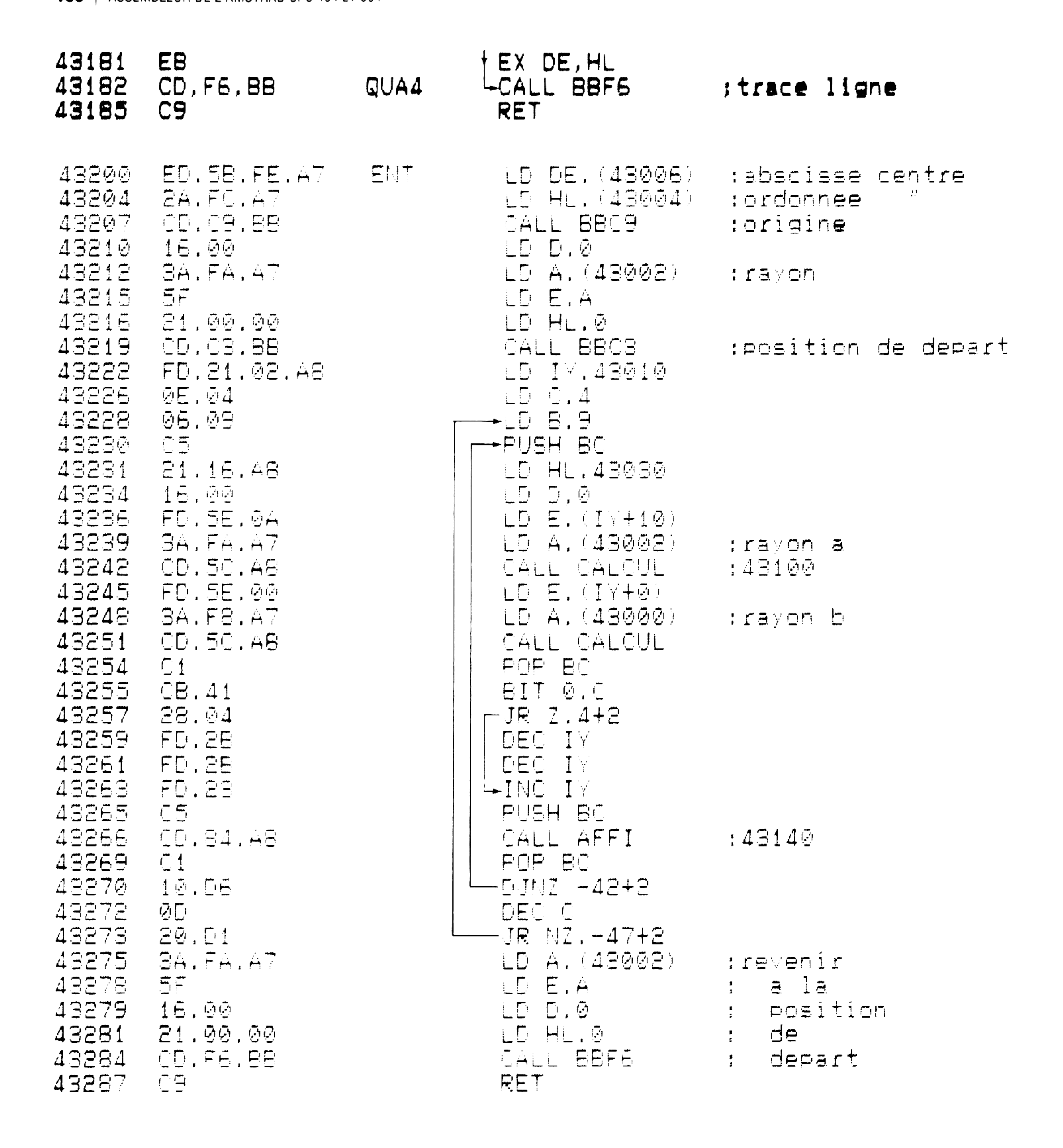
```
43000  00,00            DEF rayon ou demi-axe b:<200
43002  00,00            DEF rayon ou demi-axe a:<256
43004  00,00            DEF ordonnee centre
43006  00,00            DEF abscisse   "
43008  00,00            DEF reserve
43010  00,11,22,32,40   DEF 0.17,34,50,64: 100 x sin 0 a 40o
43015  40.57,5E.62,64   DEF 77.87,94.98,100: 100 x sin 50 a 90o
43020  64.62,5E.57,4D   DEF 100,98,94,87,77: 100 x cos 0 a 40o
43025  40,32,22,11,00   DEF 64,50,34,17,0: 100 x cos 50 a 90o
43030  00,00            DEF abscisse du point calcule
43032  00,00            DEF ordonnee          "       "
```

Entrez ensuite les trois sous-routines suivantes :

```
43100   E5              CALCUL   PUSH HL
43101   21.00.00                 LD HL.0
43104   47                       LD B.A           ;B=rayon
43105   19                       ADD HL.DE        ;HL contient
43106   10,FD                    DJNZ -3+2        :   a.100.cos
43108   01.64,00                 LD BC,100
43111   3E.00                    LD A.0
43113   ED,42                    SBC HL,BC
43115   3C                       INC A            ;n'influence pas Ci
43116   30,FB                    JR NC,-5+2
43118   3D                       DEC A            ;A contient a.cos
43119   E1                       POP HL
43120   77                       LD (HL).A        ;rangement
43121   23                       INC HL
43122   72                       LD (HL),D        :   idem
43123   23                       INC HL
43124   C9                       RET
```

```
43130   7B              COMP     LD A,E
43131   2F                       CPL
43132   5F                       LD E,A
43133   7A                       LD A,D
43134   2F                       CPL
43135   57                       LD D,A
43136   13                       INC DE
43137   C9                       RET
```

```
43140   79              AFFI     LD A,C
43141   21,16,A8                 LD HL,43030
43144   5E                       LD E,(HL)        ;DE=(43030)
43145   23                       INC HL           ;      "
43146   56                       LD D,(HL)        ;      "
43147   23                       INC HL
43148   4E                       LD C,(HL)        ;BC=(43032)
43149   23                       INC HL           ;      "
43150   46                       LD B,(HL)        ;      "
43151   69                       LD L,C
43152   60                       LD H,B           ;HL=BC
43153   FE,04                    CP 4
43155   28,19                    JR Z,QUA4        ;25+2
43157   FE,03                    CP 3
43159   28,08                    JR Z,QUA3        ;8+2
43161   FE,02                    CP 2
43163   28,09                    JR Z,QUA2        :9+2
43165   FE,01                    CP 1
43167   28,08                    JR Z,QUA1        :8+2
43169   CD,7A,A8        QUA3     CALL COMP        :43130
43172   18,08                    JR 8+2
43174   CD,7A,A8        QUA2     CALL COMP
43177   EB              QUA1     EX DE,HL
43178   CD,7A,A8                 CALL COMP
```



```
43181  EB                      EX DE,HL
43182  CD,F6,BB       QUA4     CALL BBF6        ;trace ligne
43185  C9                      RET

43200  ED,5B,FE,A7    ENT      LD DE,(43006)    :abscisse centre
43204  2A,FC,A7                LD HL,(43004)    :ordonnee    "
43207  CD,C9,BB                CALL BBC9        :origine
43210  16,00                   LD D,0
43212  3A,FA,A7                LD A,(43002)     :rayon
43215  5F                      LD E,A
43216  21,00,00                LD HL,0
43219  CD,C3,BB                CALL BBC3        :position de depart
43222  FD,21,02,A8             LD IY,43010
43226  0E,04                   LD C,4
43228  06,09                   LD B,9
43230  C5                      PUSH BC
43231  21,16,A8                LD HL,43030
43234  16,00                   LD D,0
43236  FD,5E,0A                LD E,(IY+10)
43239  3A,FA,A7                LD A,(43002)     :rayon a
43242  CD,5C,A8                CALL CALCUL      :43100
43245  FD,5E,00                LD E,(IY+0)
43248  3A,F8,A7                LD A,(43000)     :rayon b
43251  CD,5C,A8                CALL CALCUL
43254  C1                      POP BC
43255  CB,41                   BIT 0,C
43257  28,04                   JR Z,4+2
43259  FD,2B                   DEC IY
43261  FD,2B                   DEC IY
43263  FD,23                   INC IY
43265  C5                      PUSH BC
43266  CD,84,A8                CALL AFFI        :43140
43269  C1                      POP BC
43270  10,D6                   DJNZ -42+2
43272  0D                      DEC C
43273  20,D1                   JR NZ,-47+2
43275  3A,FA,A7                LD A,(43002)     :revenir
43278  5F                      LD E,A           :  a la
43279  16,00                   LD D,0           :  position
43281  21,00,00                LD HL,0          :  de
43284  CD,F6,BB                CALL BBF6        :  depart
43287  C9                      RET
```

Explication du programme

□ *Paramètres*

L'écran employé est l'écran idéal, donc d'abscisse 0-639 et d'ordonnée 0-399, quel que soit le mode d'écran.

43000 : contient le rayon (cercle), ou le demi-axe (ellipse) b qui doit être plus petit que 200.

43002 : contient le rayon ou le demi-axe a, plus petit que 256.

43004 : ordonnée du centre, entre 0 et 399, en principe ; c'est selon la valeur de b.
43006 : abscisse du centre, entre 0 et 639, en principe ; c'est selon la valeur de a.

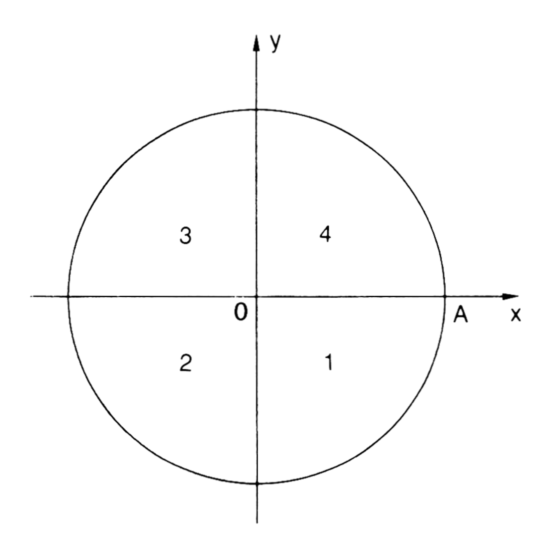
L'ordre de ces paramètres, ainsi que 2 octets par paramètre, sont nécessaires pour l'extension RSX, vue plus loin.
43010 à 43019 : est une table des sinus de 0 à 90° fois 100, en prenant les deux premiers chiffres décimaux des sinus.
43020 à 43029 : est une table des cosinus de 0 à 90° fois 100.
43030 : contient l'abscisse du point calculé : c'est a.cos t
43032 : contient l'ordonnée du point calculé : c'est b.sin t.

□ *Les sous-routines*

La sous-routine CALCUL, calcule a.cos t ou b.sin t et range le résultat dans 43030 ou 43032. A l'appel de cette sous-routine, A contient le rayon ou un demi-axe, et DE contient une valeur d'une des deux tables. Cette valeur est 100.cos t ; on multiplie d'abord par a avec résultat dans HL, puis on divise par 100, résultat dans A, soit a.cos t, que l'on place dans 43030 ; D qui reste à 0, est placé dans 43031. A l'appel suivant, on place b.sin t dans 43032 et D dans 43033.

La sous-routine COMP donne le complément vrai d'un nombre de seize bits. A l'appel, DE contient le nombre à complémenter. Au retour, DE contient le nombre complémenté. (voir partie 3).

La sous-routine AFFI trace une ligne entre deux points calculés consécutifs. A l'entrée, C contient le quadrant où l'on trace la ligne. Nous avons numéroté les quadrants de la manière ci-contre. Les valeurs des sinus et des cosinus sont les mêmes dans les quatre quadrants, au signe près.



Le début de cette routine met les valeurs correctes dans DE et HL avant d'appeler BBF6. On aménage une ou les deux valeurs selon le quadrant. N'oubliez pas qu'on calcule par rapport au centre 0.

Dans le quadrant 4, le sinus et cosinus sont positifs.
Dans le quadrant 3, les cosinus sont négatifs (avec signe).
Dans le quadrant 2, les sinus et cosinus sont négatifs.
Dans le quadrant 1, les sinus sont négatifs.

□ *La routine ENT*

La routine principale ENT trace les trente-six segments de droite pour représenter le cercle ou l'ellipse.
Les trois premières instructions fixent l'origine des coordonnées usager au centre 0.
Les instructions 43210 à 43219 fixent la position de départ A. Donc DE=a et HL=0 avant l'appel de BBC3.
En 43222, IY pointe sur le début des deux tables.
Ensuite C contrôle une boucle pour les quatre quadrants, et B contrôle une boucle qui affiche neuf segments.

La boucle B met HL à 43030, puis DE à une valeur extraite de la table des cosinus, puis A à la valeur de a ; ensuite appel de la sous-routine CALCUL ; puis DE à une valeur de la table des sinus et A à la valeur de b, et enfin, un deuxième appel de CALCUL. Les valeurs calculées sont en place pour appeler AFFI.

Il y a une petite section, de 43254 à 43265, basée sur des considérations trigonométriques, qui nous ont amenés à nous servir des mêmes tables pour les quatre quadrants. Cos 100° est égal à cos 80°, au signe près, sin 100° est égal à sin 80°, etc. Le traitement du signe se fait dans AFFI, suivant la valeur de C que l'on place dans A au début de AFFI. L'affichage du quadrant 4 se fait en progressant dans les deux tables ; l'affichage du quadrant 3 se fait en dégressant dans les deux tables, etc. Donc, dans les quadrants 4 et 2, on doit avoir INC IY et dans les quadrants 3 et 1, DEC IY. Cela s'obtient en testant le bit 0 de C. Quand C est pair, on fait INC IY et quand C est impair, on fait DEC IY.

Pour terminer, on trace le dernier segment, pour rejoindre le point de départ : 43275 à 43284.

Ce programme fonctionne dans les trois modes d'écran.
Comme vous l'avez remarqué en lisant le programme, nous n'avons fait aucun test de validité des paramètres. Faites attention aux sorties d'écran.
Pour visualiser quinze cercles concentriques, faites :

```
10 CLS
20 POKE 43004,200:POKE 43005,0:REM ordon
nee du centre:200
```

```
30 POKE 43006,64:POKE 43007,1:REM abscis
se du centre:320
40 FOR n=50 TO 190 STEP 10
50 POKE 43002,n:POKE 43000,n:REM cercle
de rayon n
60 CALL 43200
70 NEXT
80 END
```

Si vous voulez un joli dessin composé de quinze ellipses, changez la ligne 50 :

```
50 POKE 43002,n:POKE 43000,240-n
```

LE SYSTÈME RSX

Le système RSX est contrôlé par la section KERNEL du système. La routine qui introduit de nouvelles commandes est :

- *BCD1 :* Introduit une ou plusieurs commandes RSX dans le système.
  Entrée : BC contient l'adresse de la table des commandes RSX.
  HL contient l'adresse d'une zone de 4 octets, utilisés par le KERNEL.
  Sortie : D et E sont indéfinis.

  Le programme, à l'adresse donnée par BC, contient une série de sauts (autant que de commandes nouvelles), suivie d'une liste des noms de ces commandes nouvelles. La dernière lettre de chaque nom est codée par son code ASCII plus 128. Le KERNEL sait ainsi que le nom se termine. A la fin de la liste des noms, on met 0 qui est l'indicateur de la fin de la table.

  Par exemple, en supposant que adr est l'adresse donnée à BC, et qu'il y a trois nouvelles commandes :

```
adr         DEF table-nom
            JP routine1
            JP routine2
            JP routine3
table-nom   DEF L,I,G,N,E+128
            DEF C,E,R,C,L,E+128
            DEF S,O,N,N,E,Z+128
            DEF 0 indicateur fin de table
```

et le KERNEL sait que la routine 1 correspond à LIGNE, la routine 2 à CERCLE et la routine 3 à SONNEZ.

La nouvelle commande peut être suivie par un ou plusieurs paramètres. Dans ce cas le KERNEL fait pointer IX sur une zone qui contient ces paramètres, chacun sur deux octets. Cette zone a fonctionné comme une pile LIFO ; IX+0 pointe sur le dernier paramètre entré, IX+2 sur l'avant-dernier, etc.

Voyons cela avec une nouvelle commande CERCLE.

□ *Nouvelle commande CERCLE*

Entrez le programme suivant :

```
43300  01.2E,A9                LD BC,43310     ;adresse table com.
43303  21.3A.A9                LD HL.43322     ;    "    pour KERNEL
43306  CD.D1,BC                CALL BCD1       ;RSX
43309  C9                      RET
43310  33.A9                   DEF 43315       :table des noms
43312  C3.3E.A9                JP 43326        ;routine CER
43315  43.45,52.43             DEF C,E.R.C     ;nom
43319  4C,C5                   DEF L.E+128
43321  00                      DEF fin de table
43322  00,00,00.00             DEF 4 octets pour KERNEL
43326  06,08           CER     LD B.8          :transfert
43328  21.F8.A7                LD HL.43000     :  des
43331  DD,7E,00              ┌►LD A,(IX+0)     :  parametres
43334  77                    │ LD (HL),A
43335  DD,23                 │ INC IX
43337  23                    │ INC HL
43338  10.F7                 └─DJNZ -9+2
43340  C3.C0,A8                JP 43200        ;saut a ENT
```

Avant d'exploiter la nouvelle commande dans un programme BASIC, il faut faire CALL 43300 pour loger cette commande dans le système. Ensuite, la nouvelle commande doit se distinguer d'une commande BASIC de la ROM, de la façon suivante. On doit faire précéder le nom par une barre verticale, obtenue en poussant sur le a commercial shifté (SHIFT @) et faire suivre le nom par une virgule, puis par les paramètres, séparés par des virgules. Voici la syntaxe de la commande nouvelle :

| CERCLE,x,y,a,b

où x est l'abscisse du centre, y son ordonnée, a le demi-axe hozirontal et b le demi-axe vertical.

Faites la commande CALL 43300 si ce n'est déjà fait, puis :

| CERCLE,320,200,100,100

L'avantage est de pouvoir mettre cette commande n'importe où dans un programme BASIC. Les paramètres peuvent être des variables BASIC, comme le montre le programme suivant :

```
10 CALL 43300
20 CLS
30 FOR n=60 TO 560 STEP 40
40 y=60+n/2
50 |CERCLE,n,y,50,50
60 NEXT
70 END
```

Il faut absolument introduire quatre paramètres, même si on ne dessine pas d'ellipse.

Avouez que c'est plus facile d'emploi que le programme en C.M. pour lequel on devait faire une série de POKE dans le programme BASIC d'exploitation.

Il est évident que vous devez entrer tous les C.M. depuis 43000 jusqu'à 43342 pour que tout soit en place ; ou faire un LOAD si vous avez sauvé sur cassette.

Dans le bloc de sauts principal, il y a également des routines pour traiter des opérations de cassette, et des sons. Nous n'avons pas donné ces routines, car le traitement de ces sections est essentiellement temporel, et les C.M. n'apportent rien en gain de temps.

3

Déplacement de dessins multicolores

Pour terminer ce livre, nous vous proposons de construire un programme pour afficher, et effacer, un dessin multicolore, par conséquent en mode 0. En déplaçant la position d'affichage, on crée ainsi l'animation.

LE CODAGE DES ENCRES

La routine BC2C vous donne l'encre codée selon le mode d'écran. En effet, dans les trois modes, les bits d'un octet n'agissent pas de la même manière sur les couleurs (revoyez cette routine plus haut).

Pour afficher un dessin, nous chargeons successivement tous les points colorés dans la mémoire d'écran. Vous pouvez employer la routine BC1D qui donne l'adresse d'écran et le masque, puis la routine BC2C qui donne l'encre codée, puis finalement la routine BC5C pour afficher un point de couleur. L'emploi de ces trois routines dans un programme avec boucles contrôlées amène une certaine lenteur, due aux nombreux tests effectués par ces routines générales. Notre programme d'affichage exécute le codage de l'encre et l'affichage de points, toujours dans un but didactique qui a été notre souci constant lors de la rédaction de ce livre.

Voyons le codage de l'encre en mode 0. A l'affichage, un octet représente deux points. Le point de gauche est coloré par les bits 1, 5, 3, 7 et le point droit par les bits 0, 4, 2, 6 de l'octet. La mise de ces bits correspond aux couleurs d'encre de 0 à 15.

Mode 0	Bits 1,5,3,7 point gauche	Bits 0,4,2,6 point droit
0 bleu	0	0
1 jaune vif	80	40
2 turquoise vif	08	04
3 rouge vif	88	44
4 blanc brillant	20	10
5 noir	A0	50
6 bleu vif	28	14
7 magenta vif	A8	54
8 turquoise	02	01
9 jaune	82	41
10 bleu pastel	0A	05
11 rose	8A	45
12 vert vif	22	11
13 vert pastel	A2	51
14 bleu/jaune vif	2A	15
15 rose/bleu ciel	AA	55

Vous pouvez, bien sûr, changer les couleurs, mais non pas leurs numéros.

Remarquez ceci : si vous faites glisser les bits du point droit vers la gauche, avec un SLA, vous obtenez le codage du point gauche ; donc en multiplant par deux le codage du point droit, on obtient le codage du point gauche de la même couleur.

LE CODAGE DU DESSIN

Nous nous proposons d'afficher un dessin de seize points de large (donc huit octets) sur seize de haut. Pour pouvoir changer de dessin et le coder facilement, nous donnons à chaque point une valeur de 0 à 15, correspondante à la couleur du point. Il nous faut évidemment 256 octets pour représenter le dessin. Notre dessin est un visage aux yeux clignotants. Entrez ces octets :

```
43000  00,00,00,00,00,01,01,01,01,01,01,00,00,00,00,00  DEF
43016  00,00,00,01,01,01,01,01,01,01,01,01,01,00,00,00  DEF
43032  00,00,01,01,01,09,09,09,09,09,09,01,01,01,00,00  DEF
43048  00,01,01,09,09,09,09,09,09,09,09,09,09,01,01,00  DEF
43064  00,01,01,06,06,06,09,09,09,09,06,06,06,01,01,00  DEF
43080  01,01,06,02,02,02,06,09,09,06,02,02,02,05,01,01  DEF
43096  01,01,09,02,0E,02,09,09,09,09,02,0E,02,09,01,01  DEF
43112  01,01,09,09,09,09,09,09,09,09,09,09,09,09,01,01  DEF
43128  01,01,09,09,09,09,09,09,09,09,09,09,09,09,01,01  DEF
```

```
43144  01,01,09,09,09,09,05,09,09,05,09,09.09.09.01.01  DEF
43160  01,01,09,09,03,09,09,09,09,09,09,03,09.09.01.01  DEF
43176  00,01,01,09,09,03,03,04,04,03,03,09,09,01,01.00  DEF
43192  00,01,01,09,09,09,09,03,03,09,09,09,09,01,01,00  DEF
43208  00,00,01,01,09,09,09,09,09,09,09,09,01,01,00,00  DEF
43224  00,00,00,01,01,09,09,09,09,09,09,01,01,00,00,00  DEF
43240  00,00,00,00,00,01,01,01,01,01,01,00,00,00,00,00  DEF
```

Entrez ensuite les seize octets qui nous serviront pour le codage :

```
43264  00,40,04,44,10,50,14,54,01,41,05,45,11,51,15,55  DEF
```

Nous avons choisi l'adresse 43264d-A900h, car il nous suffira de charger l'octet bas avec le numéro d'encre pour pointer sur le codage d'une couleur.

PROGRAMME D'AFFICHAGE DU DESSIN MULTICOLORE

Entrez les C.M. suivants :

```
43290  64.00                  DEF abscisse entre 0 et 151
43292  64.00                  DEF ordonnee entre 16 et 199

43300  2A.1C.A9       ENT     LD HL.(43292)   :ordonnee
43303  ED.5B.1A.A9            LD DE.(43290)   :abscisse
43307  CD.1D.BC               CALL BC1D       :pos. d'ecran
43310  FD.21.F8.A7            LD IY.43000
43314  0E.10                  LD C.16         :16 lignes
43316  E5             LIGNE   PUSH HL
43317  06.08                  LD B.8          :8 octets:16 points
43319  C5             POINT   PUSH BC
43320  16.A9                  LD D.A9         :octet haut de A900
43322  FD.7E.00               LD A.(IY+0)     :saisie point
43325  5F                     LD E.A          :octet bas de A900
43326  1A                     LD A.(DE)       :encre codee
43327  CB.27                  SLA A           :point gauche
43329  47                     LD B.A          :sauve dans B
43330  FD.23                  INC IY          :point suivant
43332  FD.7E.00               LD A.(IY+0)     :saisie point
43335  5F                     LD E.A
43336  1A                     LD A.(DE)       :encre codee
43337  B0                     OR B            :unit les 2 points
43338  FD.23                  INC IY          :point suivant
43340  00                     NOP
43341  77                     LD (HL).A       :l'affiche (octet)
43342  CD.20.BC               CALL BC20       :pos. octet suivant
43345  C1                     POP BC          :boucle B
43346  10.E3                  DJNZ -29+2      :POINT
```

```
43348  E1                      POP HL
43349  CD,26,BC                CALL BC26        ;ligne inferieure
43352  0D                      DEC C
43353  20,D9                   JR NZ,-39+2      ;LIGNE
43355  C9                      RET

43360  3E,AF          EFF      LD A,AF          ;code de XOR A
43362  32,4C,A9                LD (43340),A     ;dans 43340
43365  CD,24,A9                CALL 43300       ;pour effacer
43368  AF                      XOR A            ;A=0
43369  32,4C,A9                LD (43340),A     ;NOP dans 43340
43372  C9                      RET
```

Faites MODE 0:CALL 43300 et vous voyez le dessin multicolore.
Faites CALL 43360 pour l'effacer.

Aux lignes 43320 à 43337, on code les couleurs de deux points et on les réunit dans un octet qu'on affiche à 43341. A ce moment, HL tient l'adresse d'écran. La routine BC20 donne à HL l'adresse d'écran de l'octet situé juste à droite, et ainsi de suite huit fois.

En 43348, on récupère l'adresse d'écran initiale. Puis la routine BC26 donne à HL l'adresse d'écran juste en dessous, et ainsi de suite, seize fois.

Tel quel, notre programme a un petit inconvénient : il ne traite pas séparément les abscisses de numéro impair et pair. Donc le déplacement horizontal se fait par deux points ; l'affichage à une abscisse paire puis impaire se fait au même endroit.

Pour l'effacement, on place XOR A à la place de NOP à la ligne 43340, pour que A contienne 0 pour l'affichage. Puis après CALL 43300, pour effacer aux mêmes positions, on remet NOP dans la ligne 43340.

Nous avons supposé que la couleur papier est bleue, de numéro d'encre 0, aussi bien pour l'effacement que pour l'affichage.
Le BASIC suivant vous montre une animation possible :

```
10 MODE 0
20 POKE 43291,0:POKE 43293,0:y=5
30 FOR x=50 TO 100 STEP 2
40 POKE 43290,x:POKE 43292,120+y
50 CALL 43300
60 FOR t=1 TO 100:NEXT t
70 CALL 43360
80 y=-y
90 NEXT x
100 END
```

Nous n'avons fait aucun test de validité sur les valeurs d'abscisses et d'ordonnées. Vous pouvez ajouter ces tests soit en BASIC, soit en C.M.

Vous pouvez également faire une nouvelle commande BASIC avec un RSX, pour déplacer un ou plusieurs dessins ou lutins. Vous en connaissez suffisamment.

Programmez et programmez sans cesse ; comme le microprocesseur ne se trompe pas, si une chose ne fonctionne pas dans votre programme, c'est qu'il contient une erreur. Quelle joie finale sera la vôtre quand votre programme se déroulera exactement comme vous l'aviez décidé ! Commencez, par exemple, par optimiser les programmes de ce livre, ce qui n'était pas notre but, et modifiez-les selon vos désirs.

Note sur l'Amstrad CPC 664

Cet ordinateur, mis en vente récemment, se distingue du CPC 464 par le fait qu'il exploite une disquette de 3" incorporée, et qu'il possède quelques instructions BASIC supplémentaires. Ces additions s'attribuent une zone de mémoire vive supplémentaire pour fonctionnner : la figure 1 du chapitre 6 n'est plus adéquate. Les commandes suivantes montrent cette zone :

PRINT HIMEM affiche 43903 pour le CPC 464
PRINT HIMEM affiche 42619 pour le CPC 664

Tous les programmes de ce livre sont élaborés pour prendre place aux adresses 43000 et suivantes. Les possesseurs du CPC 664 changeront le programme de chargement de la partie 1 pour que les adresses des programmes commencent à 42000, ou à 41000 ; en modifiant en conséquence, toutes les adresses absolues que les programmes comportent. Quand on dispose d'un programme Assembleur, l'emploi de directives et d'étiquettes adéquates font exécuter par l'Assembleur ce travail un peu fastidieux. Or le CPC 664 est livré avec un système d'exploitation de disquettes CP/M version 2.2 et ce CP/M est fourni avec un Assembleur. Il n'y a donc aucun problème pour les possesseurs du CPC 664 s'ils suivent le conseil que nous donnons tout au début du livre : s'initier au BASIC et à la manipulation de l'Amstrad avant d'aborder la programmation en langage machine.
Quant aux adresses des blocs de sauts, elles restent les mêmes pour les deux Amstrad.

EN CAS D'ERREUR

Tous les listings de cet ouvrage sont des originaux sortis directement de l'imprimante. Tous les programmes proposés ici ont été testés attentivement sur Amstrad CPC 464 et 664. Toute erreur serait donc exceptionnelle.

Si, malgré tout, l'un des programmes ne fonctionnait pas, assurez-vous d'avoir correctement recopié le listing en assembleur ou en code machine.

Voici quelques conseils qui vous aideront à déceler l'erreur :

- ne confondez pas 0 (zéro) et O (lettre) ;
- vérifiez les adresses hexadécimales ;
- comptez les lignes du programme à recopier et faites attention à ne pas en oublier ;
- donnez toujours le même nom à vos variables d'un bout à l'autre du programme.

Et surtout... Armez-vous de courage !

ANNEXE 1

Table 1

Liste alphabétique des mnémoniques

Leurs codes, les instructions qu'ils représentent. Le nombre d'octets nécessaires se trouve dans la colonne 2 (codes).

Mnémoniques	Codes	Instructions
A		
ADC A,R	Table 2	$A=A+R+C_i$ (C_i=indicateur C).
ADC A,(RR)	Table 2	$A=A+(RR)+C_i$.
ADC A,n	CE,nn	$A=A+n+C_i$.
ADC HL,RR	Table 3	$HL=HL+RR+C_i$.
ADD A,R	Table 2	$A=A+R$.
ADD A,(RR)	Table 2	$A=A+(RR)$.
ADD A,n	C6,nn	$A=A+n$.
ADD HL,RR	Table 3	$HL=HL+RR$.
ADD IX,RR	Table 3	$IX=IX+RR$.
ADD IY,RR	Table 3	$IY=IY+RR$.
AND R	Table 2	$A=A\wedge R$.
AND (RR)	Table 2	$A=A\wedge(RR)$.
AND n	E6,nn	$A=A\wedge n$.
B		
BIT b,R	Table 2	$Z=\overline{R}$.
BIT b,(RR)	Table 2	$Z=\overline{(RR)}_b$.

Mnémoniques	Codes	Instructions
C		
CALL NZ,pq	C4,qq,pp	si Z=0)
CALL Z,pq	CC,qq,pp	si Z=1) alors
CALL NC,pq	D4,qq,pp	si C=0) (SP−1)=PC haut.
CALL C,pq	DC,qq,pp	si C=1) (SP−2)=PC bas.
CALL PO,pq	E4,qq,pp	si P/V=0) SP=SP−2.
CALL PE,pq	EC,qq,pp	P/V=1) PC=pq.
CALL P,pq	F4,qq,pp	si S=0)
CALL M,pq	FC,qq,pp	si S=1)
		si non alors PC=PC+3.
CALL pq	CD,qq,pp	(SP−1)=PC haut, (SP−2)=PC bas, SP=SP−2 et PC=pq
CCF	3F	$C_i=\overline{C_i}$ devient le C_i précédent.
CP R	Table 2	A−R.
CP (RR)	Table 2	A−(RR).
CP n	FE,nn	A−n.
CPD	ED,A9	A−(HL) HL=HL−1 BC=BC−1
CPDR	ED,B9	A−(HL) P/V=0 si BC=0 HL=HL−1 après exécution, BC=BC−1 si non P/V=1 Répétez jusqu'à ce que BC=0 ou A=(HL)
CPI	ED,A1	A−(HL) Z=1 si A=(HL) HL=HL+1 BC=BC−1
CPIR	ED,B1	A−(HL) HL=HL+1 BC=BC−1 Répétez jusqu'à ce que BC=0 ou A=(HL)
CPL	2F	$A=\overline{A}$ (complément à 1)
D		
DAA	27	Effectue différents ajustements arithmétiques selon les indicateurs N,H et C (employé en DCB)
DEC R	Table 2	R=R−1
DEC (RR)	Table 2	(RR)=(RR)−1
DEC RR	Table 3	RR=RR−1
DI	F3	IFF=0 mise hors d'action des interruptions masquables
DJNZ e	10,ee	B=B−1 si B≠0, faire PC=PC+e+2 (e est un nombre avec signe)

Mnémoniques	Codes	Instructions
E		
EI	FB	IFF=1, remise en action des interruptions masquables
EX AF,AF'	08	A<=>A' et F<=>F' échange.
EX DE,HL	EB	DE<=>HL échange
EX (SP),RR	Table 3	(SP)<=>RRbs (SP+1)<=>RRhaut échange
EXX	D9	BC<=>BC' DE<=>DE' échange HL<=>HL'
H		
HALT	76	Suspend le microprocesseur. Celui-ci effectue des NOP jusqu'à réception d'une interruption ou d'un reset.
I		
IM 0	ED,46	Met le mode 0 d'interruption, le dispositif envoie une instruction.
IM 1	ED,56	Met le mode 1 d'interruption, un RST 38h sera exécuté.
IM 2	ED,5E	Met le mode 2 d'interruption, le dispositif envoie un octet considéré comme octet bas d'une adresse. Son octet haut est pris dans I.
IN R, (C)	Table 2	R=(C) Le contenu de C fournit l'octet bas de l'adresse du port. B fournit l'octet haut.
IN A,(p)	DB,pp	A=(p) p est l'octet bas de l'adresse du port. A fournit l'octet haut.
INC R	Table 2	R=R+1
INC (RR)	Table 2	(RR)=(RR)+1
INC RR	Table 3	RR=RR+1
IND	ED,AA	(HL)=(C) Z=1 si B=0 B=B−1 après exécution, HL=HL−1 sinon Z=0 Entrée du port adressé par C avec DEC.
INDR	ED,BA	(HL)=(C), B=B−1, HL=HL−1. Répétez jusqu'à ce que B=0. Entrée de bloc du port adressé par C avec DEC.
INI	ED,A2	(HL)=(C) Z=1 si B=0 B=B−1 après exécution HL=HL+1 sinon Z=0 Entrée du port adressé par C avec INC.

Mnémoniques	Codes	Instructions
INIR	ED,B2	(HL)=(C) B=B−1 HL=HL+1 Répétez jusqu'à ce que B=0. Entrée de bloc du port adressé par C avec INC.
J		
JP NZ,pq	C2,qq,pp	si Z=0 } alors PC=pq
JP Z,pq	CA,qq,pp	si Z=1
JP NC,pq	D2,qq,pp	si C=0
JP C,pq	DA,qq,pp	si C=1
JP P0,pq	E2,qq,pp	si P/V/=0
JP PE,pq	EA,qq,pp	si P/V=1
JP P,pq	F2,qq,pp	si S=0
JP M,pq	FA,qq,pp	si S=1 si non, alors PC=PC+3
JP pq	C3,qq,pp	PC=pq } alors PC=PC+e+2
JP RR	Table 3	PC=RR
JR NZ,e	20,ee	si Z=0
JR Z,e	28,ee	si Z=1
JR NC,e	30,ee	si C=0
JR C,e	38,ee	si C=1 si non PC=PC+2 (e est un nombre avec signe).
JR e	18,ee	PC=PC+e+2
L		
LD RR,(nm)	Table 3	RRbas=(nm) RRhaut=(nm+1)
LD (nm),RR	Table 3	(nm)=RRbas (nm+1)=RRhaut
LD RR,nm	Table 3	RR=nm
LD SP,RR	Table 3	SP=RR
LD R,n	Table 2	R=n
LD (RR),n	Table 2	(RR)=n
LD A,R ou (RR)	Table 2	A=R ou A=(RR)
LD B,R ou (RR)	Table 2	B=R ou B=(RR)
LD C,R ou (RR)	Table 2	C=R ou C=(RR)
LD D,R ou (RR)	Table 2	D=R ou D=(RR)
LD E,R ou (RR)	Table 2	E=R ou E=(RR)
LD H,R ou (RR)	Table 2	H=R ou H=(RR)
LD L,R ou (RR)	Table 2	L=R ou L=(RR)
LD (HL),R	Table 2	(HL)=R
LD (IX+d),R	Table 2	(IX+d)=R
LD (IY+d),R	Table 2	(IY+d)=R
LD (BC),A	02	(BC)=A
LD (DE),A	12	(DE)=A

Mnémoniques	Codes	Instructions
LD (nm),A	32,mm,nn	(nm)=A
LD A,(BC)	0A	A=(BC)
LD A,(DE)	1A	A=(DE)
LD A,(nm)	3A,mn,nn	A=(nm)
LD I,A	ED,47	I=A
LD R,A	ED,4F	R=A
LD A,I	ED,57	A=I P/V=IFF2
LD A,R	ED,5F	A=R
LDD	ED,A8	(DE)=(HL) DE=DE−1 HL=HL−1 BC=BC−1 P/V=0 si BC=0 après exécution
LDI	ED,A0	(DE)=(HL) sinon P/V=1 DE=DE+1 HL=HL+1 BC=BC−1 Chargement d'octet avec DEC ou INC.
LDDR	ED,B8	(DE)=(HL) DE=DE−1 HL=HL−1 BC=BC−1 Répétez jusqu'à ce que BC=0
LDIR	ED,B0	(DE)=(HL) DE=DE+1 HL=HL+1 BC=BC−1 Répétez jusqu'à ce que BC=0 chargement de bloc avec DEC ou INC.
N		
NEG	ED,44	A=0−A (complément à 2)
NOP	00	Pas d'opération.
O		
OR R	Table 2	A=A V R
OR (RR)	Table 2	A=A V (RR)
OR n	F6,nn	A=A V n
OTDR	ED,BB	(C)=(HL) B=B−1 HL=HL−1 Répétez jusqu'à ce que B=0
OTIR	ED,B3	(C)=(HL) B=B−1 HL=HL+1 Répétez jusqu'à ce que B=0. Sortie de bloc avec DEC ou INC. C fournit l'octet bas de l'adresse du port

Mnémoniques	Codes	Instructions
		B après décrémentation fournit l'octet haut.
OUTD	ED,AB	(C)=(HL)
		B=B−1
		HL=HL−1 Z=1 si B=0 après exécution
		sinon Z=0
OUTI	ED,A3	(C)=(HL)
		B=B−1
		HL=HL+1
		Sortie d'octet avec DEC ou INC.
OUT (C),R	Table 2	(C)=R
		R est mis dans le port adressé par C pour l'octet bas et par B pour le haut.
OUT (p),A	D3,pp	(p)=A
		A est mis dans le port adressé par p.
P		
POP RR	Table 3	RRbas =(SP)
		RRhaut =(SP+1)
		SP=SP+2
PUSH RR	Table 3	(SP−1)=RRhaut
		(SP−2)=RRbas
		SP=SP−2
R		
RES b,R	Table 2	R_b=0
RES b(RR)	Table 2	$(RR)_b$=0
RET	C9	PCbas=(SP)
		PChaut=(SP+1)
		SP=SP+2
RETI	ED,4D	PCbas=(SP) retour d'interruption
		PChaut=(SP+1) EI doit être avant RETI
		SP=SP+2
RETN	ED,45	PCbas=(SP) retour d'interruption
		PChaut=(SP+1) non masquable
		SP=SP+2
		IFF1=IFF2
RET NZ	C0	si Z=0
RET Z	C8	si Z=1
RET NC	D0	si C=0
RET C	D8	si C=1 } alors PCbas=(SP)
RET P0	E0	si P/V=0 } PChaut=(SP+1)
RET PE	E8	si P/V=1 } SP=SP+2
RET P	F0	si S=0
RET M	F8	si S=1
		sinon alors PC=PC+1

Mnémoniques	Codes	Instructions
RLA	17	C_i ← [7 ← 0] A
RL R ou (RR)	Table 2	C_i ← [7 ← 0] R ou (RR)
RLCA	07	C_i ← [7 ← 0] A
RLC R ou (RR)	Table 2	C_i ← [7 ← 0] R ou (RR)
RLD	ED,6F	A [7 4 | 3 0] [7 4 | 3 0] (HL) Rotation à gauche décimale (par quartet - pour DCB)
RRA	1F	[7 → 0] A → C_i
RR R ou (RR)	Table 2	[7 → 0] R ou (RR) → C_i
RRCA	OF	[7 → 0] A → C_i
RRC R ou (RR)	Table 2	[7 → 0] R ou (RR) → C_i
RRD	ED,67	A [7 4 | 3 0] [7 4 | 3 0] (HL) Rotation à droite décimale (par quartet - pour DCB)
RST 00	C7	p=00
RST 08	CF	p=08
RST 10	D7	p=10h (SP−1)=PChaut
RST 18	DF	p=18h (SP−2)=PCbas
RST 20	E7	p=20h SP=SP−2
RST 28	EF	p=28h PChaut=0
RST 30	F7	p=30h PCbas=p
RST 38	FF	p=38h
S		
SUB A,R	Table 2	A=A−R
SUB A,(RR)	Table 2	A=A−(RR)

Mnémoniques	Codes	Instructions
SUB A,n	D6,nn	A=A−n
SBC A,R	Table 2	A=A−R−C_i
SBC A,(RR)	Table 2	A=A−(RR)−C_i
SBC A,n	DE,nn	A=A−n−C_i
SBC HL,RR	Table 3	HL=HL−RR−C_i
SCF	37	C_i=1
SET b,R	Table 2	R_b=1
SET b,(RR)	Table 2	$(RR)_b$=1
SLA R ou (RR)	Table 2	C_i ← [7 ← 0] ← 0 R ou (RR) Glissement à gauche arithmétique
SRA R ou (RR)	Table 2	[7 → 0] → C_i Glissement à droite arithmétique
SRL R ou (RR)	Table 2	0 → [7 → 0] → C_i Glissement à droite logique
X		
XOR R	Table 2	A=A∀ R
XOR (RR)	Table 2	A=A∀ (RR)
XOR n	EE,nn	A=A∀ n

Table 2

Liste alphabétique des mnémoniques avec leurs

Mnémoniques	R			
	A	B	C	D
A				
ADC A,R	8F	88	89	8A
ADC A,(RR)				
ADD A,R	87	80	81	82
ADD A,(RR)				
AND R	A7	A0	A1	A2
AND (RR)				
B				
BIT 0,R ou (RR)	CB,47	CB,40	CB,41	CB,42
BIT 1,R ou (RR)	CB,4F	CB,48	CB,49	CB,4A
BIT 2,R ou (RR)	CB,57	CB,50	CB,51	CB,52
BIT 3,R ou (RR)	CB,5F	CB,58	CB,59	CB,5A
BIT 4,R ou (RR)	CB,67	CB,60	CB,61	CB,62
BIT 5,R ou (RR)	CB,6F	CB,68	CB,69	CB,6A
BIT 6,R ou (RR)	CB,77	CB,70	CB,71	CB,72
BIT 7,R ou (RR)	CB,7F	CB,78	CB,79	CB,7A
C				
CP R	BF	B8	B9	BA
CP (RR)				
D				
DEC R	3D	05	0D	15
DEC (RR)				
I				
IN R,(C)	ED,78	ED,40	ED,48	ED,50
INC R	3C	04	0C	14
INC (RR)				
L				
LD R,n	3E,nn	06,nn	0E,nn	16,nn
LD (RR),n				
LD A,R ou (RR)	7F	78	79	7A
LD B,R ou (RR)	47	40	41	42
LD C,R, ou (RR)	4F	48	49	4A
LD D,R, ou (RR)	57	50	51	52
LD E,R ou (RR)	5F	58	59	5A

codes qui montrent le nombre d'octets nécessaires

			(RR)		
E	H	L	(HL)	(IX+d)	(IY+d)
8B	8C	8D	8E	DD,8E,dd	FD,8E,dd
83	84	85	86	DD,86,DD	FD,86,dd
A3	A4	A5	A6	DD,A6,dd	FD,A6,dd
CB,43	CB,44	CB,45	CB,46	DD,CB,dd,46	FD,CB,dd,46
CB,4B	CB,4C	CB,4D	CB,4E	DD,CB,dd,4E	FD,CB,dd,4E
CB,53	CB,54	CB,55	CB,56	DD,CB,dd,56	FD,CB,dd,56
CB,5B	CB,5C	CB,5D	CB,5E	DD,CB,dd,5E	FD,CB,dd,5E
CB,63	CB,64	CB,65	CB,66	DD,CB,dd,66	FD,CB,dd,66
CB,6B	CB,6C	CB,6D	CB,6E	DD,CB,dd,6E	FD,CB,dd,6E
CB,73	CB,74	CB,75	CB,76	DD,CB,dd,76	FD,CB,dd,76
CB,7B	CB,7C	CB,7D	CB,7E	DD,CB,dd,7E	FD,CB,dd,7E
BB	BC	BD	BE	DD,BE,dd	FD,BE,dd
1D	25	2D	35	DD,35,dd	FD,35,dd
ED,58	ED,60	ED,68			
1C	24	2C	34	DD,34,dd	FD,34,dd
1E,nn	26,nn	2E,nn	36,nn	DD,36,dd,nn	FD,36,dd,nn
7B	7C	7D	7E	DD,7E,dd	FD,7E,dd
43	44	45	46	DD,46,dd	FD,46,dd
4B	4C	4D	4E	DD,4E,dd	FD,4E,dd
53	54	55	56	DD,56,dd	FD,56,dd
5B	5C	5D	5E	DD,5E,dd	FD,5E,dd

Mnémoniques	R			
	A	B	C	D
LD H,R ou (RR)	67	60	61	62
LD L,R ou (RR)	6F	68	69	6A
LD (HL),R	77	70	71	72
LD (IX+d),R	DD,77,dd	DD,70,dd	DD,71,dd	DD,72,dd
LD (IY+d),R	FD,77,dd	FD,70,dd	FD,71,dd	FD,72,dd
O				
OR R	B7	B0	B1	B2
OR (RR)				
OUT (C),R	ED,79	ED,41	ED,49	ED,51
R				
RES 0,R ou (RR)	CB,87	CB,80	CB,81	CB,82
RES 1,R ou (RR)	CB,8F	CB,88	CB,89	CB,8A
RES 2,R ou (RR)	CB,97	CB,90	CB,91	CB,92
RES 3,R ou (RR)	CB,9F	CB,98	CB,99	CB,9A
RES 4,R ou (RR)	CB,A7	CB,A0	CB,A1	CB,A2
RES 5,R ou (RR)	CB,AF	CB,A8	CB,A9	CB,AA
RES 6,R ou (RR)	CB,B7	CB,B0	CB,B1	CB,B2
RES 7,R ou (RR)	CB,BF	CB,B8	CB,B9	CB,BA
RL R ou (RR)	CB,17	CB,10	CB,11	CB,12
RLC R ou (RR)	CB,07	CB,00	CB,01	CB,02
RR R ou (RR)	CB,1F	CB,18	CB,19	CB,1A
RRC R ou (RR)	CB,0F	CB,08	CB,09	CB,0A
S				
SUB A,R	97	90	91	92
SUB A,(RR)				
SBC A,R	9F	98	99	9A
SBC A,(RR)				
SET 0,R ou (RR)	CB,C7	CB,C0	CB,C1	CB,C2
SET 1,R ou(RR)	CB,CF	CB,C8	CB,C9	CB,CA
SET 2,R ou (RR)	CB,D7	CB,D0	CB,D1	CB,D2
SET 3,R ou (RR)	CB,DF	CB,D8	CB,D9	CB,DA
SET 4,R ou (RR)	CB,E7	CB,E0	CB,E1	CB,E2
SET 5,R ou (RR)	CB,EF	CB,E8	CB,E9	CB,EA
SET 6,R ou (RR)	CB,F7	CB,F0	CB,F1	CB,F2
SET 7,R ou (RR)	CB,FF	CB,F8	CB,F9	CB,FA
SLA R ou (RR)	CB,27	CB,20	CB,21	CB,22
SRA R ou (RR)	CB,2F	CB,28	CB,29	CB,2A
SRL R ou (RR)	CB,3F	CB,38	CB,39	CB,3A
X				
XOR R	AF	A8	A9	AA
XOR (RR)				

			(RR)		
E	H	L	(HL)	(IX+d)	(IY+d)
63	64	65	66	DD,66,dd	FD,66,dd
6B	6C	6D	6E	DD,6E,dd	FD,6E,dd
73	74	75			
DD,73,dd	DD,74,dd	DD,75,dd			
FD,73,dd	FD,74,dd	FD,75,dd			
B3	B4	B5			
			B6	DD,B6,dd	FD,B6,dd
ED,59	ED,61	ED,69			
CB,83	CB,84	CB,85	CB,86	DD,CB,dd,86	FD,CB,dd,86
CB,8B	CB,8C	CB,8D	CB,8E	DD,CB,dd,8E	FD,CB,dd,8E
CB,93	CB,94	CB,95	CB,96	DD,CB,dd,96	FD,CB,dd,96
CB,9B	CB,9C	CB,9D	CB,9E	DD,CB,dd,9E	FD,CB,dd,9E
CB,A3	CB,A4	CB,A5	CB,A6	DD,CB,dd,A6	FD,CB,dd,A6
CB,AB	CB,AC	CB,AD	CB,AE	DD,CB,dd,AE	FD,CB,dd,AE
CB,B3	CB,B4	CB,B5	CB,B6	DD,CB,dd,B6	FD,CB,dd,B6
CB,BB	CB,BC	CB,BD	CB,BE	DD,CB,dd,BE	FD,CB,dd,BE
CB,13	CB,14	CB,15	CB,16	DD,CB,dd,16	FD,CB,dd,16
CB,03	CB,04	CB,05	CB,06	DD,CB,dd,06	FD,CB,dd,06
CB,1B	CB,1C	CB,1C	CB,1E	DD,CB,dd,1E	FD,CB,dd,1E
CB,0B	CB,0C	CB,0D	CB,0E	DD,CB,dd,0E	FD,CB,dd,0E
93	94	95			
			96	DD,96,dd	FD,96,dd
9B	9C	9D			
			9E	DD,9E,dd	FD,9E,dd
CB,C3	CB,C4	CB,C5	CB,C6	DD,CB,dd,C6	FD,CB,dd,C6
CB,CB	CB,CC	CB,CD	CB,CE	DD,CB,dd,CE	FD,CB,dd,CE
CB,D3	CB,D4	CB,D5	CB,D6	DD,CB,dd,D6	FD,CB,dd,D6
CB,DB	CB,DC	CB,DD	CB,DE	DD,CB,dd,DE	FD,CB,dd,DE
CB,E3	CB,E4	CB,E5	CB,E6	DD,CB,dd,E6	FD,CB,dd,E6
CB,EB	CB,EC	CB,ED	CB,EE	DD,CB,DD,EE	FD,CB,dd,EE
CB,F3	CB,F4	CB,F5	CB,F6	DD,CB,dd,F6	FD,CB,dd,F6
CB,FB	CB,FC	CB,FD	CB,FE	DD,CB,dd,FE	FD,CB,dd,FE
CB,23	CB,24	CB,25	CB,26	DD,CB,dd,26	FD,CB,dd,26
CB,2B	CB,2C	CB,2D	CB,2E	DD,CB,dd,2E	FD,CB,dd,2E
CB,3B	CB,3C	CB,3D	CB,3E	DD,CB,dd,3E	FD,CB,dd,3E
AB	AC	AD			
			AE	DD,AE,dd	FD,AE,dd

Table 3

Liste alphabétique des mnémoniques avec leurs codes qui montrent le nombre d'octets nécessaires

Mnémo-niques	RR						
	AF	BC	DE	HL	IX	IY	SP
ADC HL,RR		ED,4A	ED,5A	ED,6A			ED,7A
ADD HL,RR		09	19	29			39
ADD IX,RR		DD,09	DD,19		DD,29		DD,39
ADD IY,RR		FD,09	FD,19			FD,29	FD,39
DEC RR		0B	1B	2B	DD,2B	FD,2B	3B
EX (SP),RR				E3	DD,E3	FD,E3	
INC RR		03	13	23	DD,23	FD,23	33
JP RR				E9	DD,E9	FD,E9	
LD RR,(nm)		ED,4B,mm,nn	ED,5B,mm,nn	ED,6B,mm,nn			ED,7B,mm,nn
LD RR,(nm)				2A,mm,nn	DD,2A,mm,nn	FD,2A,mm,nn	
LD (nm),RR		ED,43,mm,nn	ED,53,mm,nn	ED,63,mm,nn			ED,73,mm,nn
LD (nm),RR				22,mm,nn	DD,22,mm,nn	FD,22,mm,nn	
LD RR,nm		01,mm,nn	11,mm,nn	21,mm,nn	DD,21,mm,nn	FD,21,mm,nn	31,mm,nn
LD SP,RR				F9	DD,F9	FD,F9	
POP RR	F1	C1	D1	E1	DD,E1	FD,E1	
PUSH RR	F5	C5	D5	E5	DD,E5	FD,E5	
SBC HL,RR		ED,42	ED,52	ED,62			ED,72

Table 4

Liste des mnémoniques ayant de l'influence sur les quatre indicateurs principaux

	S	Z	P/V	C
ADC ; ADD A,X ; CP ; EX AF,A'F' ;	x	x	x	x
NEG ; RL R ; RLC R ; RR R ; RRC R ;	x	x	x	x
SUB ; SBC ; SLA R ; SRA R ; SRL R	x	x	x	x
AND ; OR ; XOR	x	x	x	0
ADD HL (ou IX ou IY),RR ; CCF				x
RLA ; RLCA ; RRA ; RRCA				x
SCF				1
LDDR ; LDIR			0	
LDD ; LDI			(1)	
LD A,I ; LD A,R	x	x		
BIT		x		
DEC R ou (RR) ; INC R ou (RR) ; IN R,(C)	x	x	x	
CPD ; CPDR ; CPI ; CPIR	x	(1)	(2)	
INDR ; INIR ; OTDR ; OTIR		1		
IND ; INI ; OUTD ; OUTI		(3)		

(1) : P/V = 0 si BC = 0 après exécution ; sinon P/V = 1
(2) : Z = 1 si A = (HL)
(3) : Z = 1 si B = 0 après exécution ; sinon Z = 0

ANNEXE 2

Table des nombres avec signe pour les valeurs négatives

h	d comp. à 2	h	d comp. à 2	h	d comp. à 2	h	d comp. à 2
80	−128	A0	−96	C0	−64	E0	−32
81	−127	A1	−95	C1	−63	E1	−31
82	−126	A2	−94	C2	−62	E2	−30
83	−125	A3	−93	C3	−61	E3	−29
84	−124	A4	−92	C4	−60	E4	−28
85	−123	A5	−91	C5	−59	E5	−27
86	−122	A6	−90	C6	−58	E6	−26
87	−121	A7	−89	C7	−57	E7	−25
88	−120	A8	−88	C8	−56	E8	−24
89	−119	A9	−87	C9	−55	E9	−23
8A	−118	AA	−86	CA	−54	EA	−22
8B	−117	AB	−85	CB	−53	EB	−21
8C	−116	AC	−84	CC	−52	EC	−20
8D	−115	AD	−83	CD	−51	ED	−19
8E	−114	AE	−82	CE	−50	EE	−18
8F	−113	AF	−81	CF	−49	EF	−17
90	−112	B0	−80	D0	−48	F0	−16
91	−111	B1	−79	D1	−47	F1	−15
92	−110	B2	−78	D2	−46	F2	−14
93	−109	B3	−77	D3	−45	F3	−13
94	−108	B4	−76	D4	−44	F4	−12
95	−107	B5	−75	D5	−43	F5	−11
96	−106	B6	−74	D6	−42	F6	−10
97	−105	B7	−73	D7	−41	F7	−9
98	−104	B8	−72	D8	−40	F8	−8
99	−103	B9	−71	D9	−39	F9	−7
9A	−102	BA	−70	DA	−38	FA	−6
9B	−101	BB	−69	DB	−37	FB	−5
9C	−100	BC	−68	DC	−36	FC	−4
9D	−99	BD	−67	DD	−35	FD	−3
9E	−98	BE	−66	DE	−34	FE	−2
9F	−97	BF	−65	DF	−33	FF	−1

ANNEXE 3

Conversion hexadécimal - décimal

Chiffre hexa.	Poids 16^3	Poids 16^2	Poids 16^1	Poids 16^0
0	0	0	0	0
1	4096	256	16	1
2	8192	512	32	2
3	12288	768	48	3
4	16384	1024	64	4
5	20480	1280	80	5
6	24576	1536	96	6
7	28672	1792	112	7
8	32768	2048	128	8
9	36864	2304	144	9
A	40960	2560	160	10
B	45056	2816	176	11
C	49152	3072	192	12
D	53248	3328	208	13
E	57344	3584	224	14
F	61440	3840	240	15

INDEX

Liste alphabétique des mnémoniques du microprocesseur Z 80, et les pages où elles sont expliquées.

CONSEILS DE LECTURE

Pour approfondir vos connaissances en BASIC Amstrad et mieux connaître le système des CPC 464 et 664, P.S.I. vous propose une palette d'ouvrages utiles.

Pour maîtriser le BASIC Amstrad

□ **BASIC Amstrad 1 - Méthode pratiques** - Jacques Boisgontier et Bruno Césard (Editions du P.S.I.)
2 - Programme *(à paraître)* - Jacques Boisgontier (Editions du P.S.I.)

Pour tous ceux qui ont déjà pratiqué un BASIC, voici deux ouvrages de perfectionnement du BASIC Amstrad.

□ **BASIC plus 80 routines sur Amstrad** - Michel Martin (Editions du P.S.I.)

Pour pousser votre Amstrad au maximum de ses capacités : 80 routines de simulation d'instructions qui n'existent pas en BASIC Amstrad.

Pour programmer en langage machine

□ **Clefs pour Amstrad** - Daniel Martin (Editions du P.S.I.)

Mémento présentant synthétiquement le jeu d'instructions du Z 80, les points d'entrée des routines système, les connecteurs et brochages, etc. Le livre de chevet du programmeur sur Amstrad.

□ **Programmer en assembleur** - Alain Pinaud (Editions du P.S.I.)

Une introduction complète à la programmation du Z 80 pour toutes les machines dotées de ce microprocesseur.

Achevé d'imprimer en septembre 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : septembre 1985

N° d'impression : 508022
N° d'édition : 86595-295-1
ISBN : 2-86595-295-9

Votre avis nous intéresse

- Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques sur le présent livre.
- Si vous souhaitez des éclaircissements techniques, écrivez-nous, nous adresserons votre demande à l'auteur qui ne manquera pas de vous répondre directement.

- Ce livre vous donne-t-il toute satisfaction?

- Y a-t-il un aspect du problème que vous auriez aimé voir abordé?

Comment avez-vous eu connaissance de ce livre?

- ☐ publicité
- ☐ catalogue
- ☐ boutique micro
- ☐ autres
- ☐ cadeau
- ☐ librairie
- ☐ exposition

Avez-vous déjà acquis des livres PSI?

lesquels?

qu'en pensez-vous?

Nom ______ Prénom ______ Age

Adresse

Profession

Centre d'intérêt



ASSEMBLEUR DE L'AMSTRAD

CPC 464 ET 664

"**A**ssembleur de l'Amstrad" s'adresse à vous, possesseurs de CPC 464 et 664, qui avez une bonne pratique du BASIC et souhaitez programmer votre Amstrad en langue machine.
Vous vous initierez, dans un premier temps, aux principes de base de l'Assembleur du Z80 pour appliquer vos connaissances, dans un deuxième temps, aux particularités de l'Amstrad, notamment le générateur de sons.

Le jeu d'instructions du Z80 vous permettra d'obtenir rapidité d'exécution et économie de mémoire, tandis que des routines et adresses utiles vous apprendront à utiliser à fond les périphériques des Amstrad CPC 464 et 664.
De nombreux exemples et exercices vous aideront à programmer en code machine aussi facilement qu'en BASIC.



PHOTO L. MAUNOURY

ÉDITIONS DU P.S.I.
BP 86 - 77402 LAGNY S/MARNE CEDEX - FRANCE

ISBN 2-86595-295-9 **105 FF**


ASSEMBLEUR DE L'AMSTRAD
P.S.I.



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/